

Lawrence Berkeley National Laboratory

LBL Publications

Title

Learning continuous models for continuous physics

Permalink

<https://escholarship.org/uc/item/0tv7c461>

Journal

Communications Physics, 6(1)

ISSN

2399-3650

Authors

Krishnapriyan, Aditi S
Queiruga, Alejandro F
Erichson, N Benjamin
[et al.](#)

Publication Date

2023

DOI




10.1038/s42005-023-01433-4

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed

Learning continuous models for continuous physics

Aditi S. Krishnapriyan ^{1,2}, Alejandro F. Queiruga³, N. Benjamin Erichson ^{2,4} & Michael W. Mahoney^{1,2,4}

Dynamical systems that evolve continuously over time are ubiquitous throughout science and engineering. Machine learning (ML) provides data-driven approaches to model and predict the dynamics of such systems. A core issue with this approach is that ML models are typically trained on discrete data, using ML methodologies that are not aware of underlying continuity properties. This results in models that often do not capture any underlying continuous dynamics—either of the system of interest, or indeed of any related system. To address this challenge, we develop a convergence test based on numerical analysis theory. Our test verifies whether a model has learned a function that accurately approximates an underlying continuous dynamics. Models that fail this test fail to capture relevant dynamics, rendering them of limited utility for many scientific prediction tasks; while models that pass this test enable both better interpolation and better extrapolation in multiple ways. Our results illustrate how principled numerical analysis methods can be coupled with existing ML training/testing methodologies to validate models for science and engineering applications.

¹University of California, Berkeley, Berkeley, CA, USA. ²Lawrence Berkeley National Laboratory, Berkeley, CA, USA. ³Google Research, Mountain View, CA, USA. ⁴International Computer Science Institute, Berkeley, CA, USA. ✉email: aditik1@berkeley.edu

Dynamical systems—systems whose state varies over time—describe many chemical, physical, and biological processes. Thus, understanding and describing these dynamical systems is important for many scientific and engineering applications. Dynamical systems can often be described by *differential* equations which evolve continuously in time, meaning that the domain of the solution spans a continuum¹. In such systems, the gap between any two timesteps can be subdivided into an infinite number of infinitely smaller timesteps. In practice, these systems are often identified via a finite set of discrete observational data, and there is a long history within scientific computing for dealing with this discrete-to-continuous gap: experimentally measuring scientific data at sufficiently fine timescales to resolve approximately-continuous dynamics of interest; formulating theory within function spaces of sufficient smoothness to guarantee certain continuity requirements; and developing numerical algorithms that come with appropriate stability and convergence guarantees.

Machine learning (ML) techniques have recently been shown to provide a powerful approach to model and learn from discrete data, and many scientific fields make extensive use of data-driven methods for describing^{2–4}, discovering^{5–7}, identifying^{8, 9}, predicting^{10–17}, and controlling^{2, 18–21} dynamics. These approaches (see ref. 22 for a survey) include purely data-driven methods that learn from observational data points²³, adding constraints to ML methods that aim to respect the relevant physics²⁴, and/or hybrid methods combining classical numerical solvers with (say) deep learning^{25, 26}.

In many scientific and engineering applications, we observe measurements that yield a series of discrete data points $\{x_0, x_1, x_2, \dots, x_N\}$, where each point is spaced apart by some timestep size Δt . There are many techniques from ML and statistical data analysis to learn data-driven input-output mappings ($G: x_n \rightarrow x_{n+1}$) that can provide an approximation for the next discrete timestep. One popular class of data-driven input-output mappings is given by neural networks (NNs). A NN, denoted as \mathcal{N} , can be trained to predict x_{n+1} from x_n by learning model parameters θ :

$$x_{n+1} = \mathcal{N}(x_n; \theta). \quad (1)$$

However, when considering continuous dynamical systems, there are challenges with this approach. Most obviously, this approach does not learn a continuous function^{27–30}; it simply learns a function that predicts subsequent discrete time steps. This is to be expected, as this model is optimized to make (discrete) point estimates, i.e., to predict solutions at specific (discrete) points. For this reason, predicting future states of a dynamical system with this approach can result in compounding errors of the dynamics over time^{19, 31}.

A related approach is to assume that the discrete data points can be modeled and described by a continuous differential equation of the form,

$$\frac{dx(t)}{dt} = F(x(t)), \quad (2)$$

where F is a function that describes the vector field. In some cases, there is an underlying true F , while in other cases it is simply a modeling assumption. A challenge is that we cannot derive F from first-principles in many situations. Instead, we can use a data-driven approach for modeling F . For instance, an arbitrary NN architecture \mathcal{N} can be used to model the vector field F ,

$$\frac{dx(t)}{dt} = \mathcal{N}(x(t); \theta). \quad (3)$$

This approach, so-called Neural Ordinary Differential Equations (ODE-Nets), has been proposed to model temporal systems^{32–38}.

It is often assumed or simply taken for granted that ODE-Nets and other ML methods for ODEs automatically capture some continuous dynamics, either of the system that generated the data or of some related system^{39–43}.

However, due to how ODE-Nets are trained, i.e., to predict solutions at specific (discrete) points, these models can easily fail to learn even the simplest continuous dynamical systems^{44, 45}, even when they accurately fit the temporal discretization (i.e., the discrete training points and testing points). An ODE-Net that incorrectly learns a continuous model will simply provide high-quality discrete time predictions — i.e., it is not a ContinuousNet but is simply a very good DiscreteNet⁴⁴.

Such a model will fail to extrapolate to new data points outside the temporal discretization, and it will fail to interpolate the solution at timesteps in between the discrete training data. It can also fail to correctly identify qualitative long-term behavior such as bifurcations⁴⁶. As we demonstrate later, this Discrete-versus-Continuous distinction affects non-NN ML methods as well, even when they accurately fit the temporal discretization of the data.

Figure 1 illustrates the difference between a model that has learned to predict discrete data points and a model that has learned an underlying continuous dynamics. After training a model at a given discretization Δt , the trained model can be used to predict trajectories at arbitrary timestep sizes h during inference. For validation, an error metric $Error(h)$ can be defined over a holdout trajectory that allows for evaluation with discretizations that are different than the data spacing. Learning a discrete-only model (a DiscreteNet) means that only the discrete training points—and potentially testing points at the same discretization (i.e., when $h = \Delta t$)—are learned. When evaluating using $Error(h = \Delta t)$, the model will appear to perform well. The model may perform well on testing points with a similar discretization, but it will perform poorly for points sampled with other discretizations; that is, $Error(h \neq \Delta t)$ can be much worse than a discrete-only testing methodology would determine. This will even occur when the discretization $h \rightarrow 0$, counter to expectations. In contrast, learning a meaningfully continuous model means that the model can converge to a smooth solution as the discretization $h \rightarrow 0$, or at least that its error will decrease gradually and level off as $h \rightarrow 0$. (This will be true regardless of whether the learned continuous model corresponds to the true underlying model, even assuming that such a true model exists and/or is well-conditioned.) In this case, the model will perform well for a broader range of temporal discretizations and thus have a better approximation of the continuous dynamical system.

In this work, we adapt methods from numerical analysis theory to develop a methodology to verify whether an ML model has learned a meaningfully continuous function that describes a dynamical system of interest. Specifically, we introduce a modified convergence test to verify and validate whether a model has learned continuous dynamics for a physical system. Our method allows us to verify that a model approximates a continuous differential operator, rather than only learning discrete points at a given temporal discretization, in the same sense that discrete algorithms from numerical analysis can be said to approximate continuous functions. We also introduce the notion of a ContinuousNet to refer to an ODE-Net model that exhibits the convergence properties that are expected for a continuous time system:

Definition 1. (ContinuousNet). An ODE-Net,

$$\frac{dx(t)}{dt} = \mathcal{N}_\theta(x(t)), \quad x(0) = x_0 \in \mathbb{R}^n, \quad \text{and } t \in \mathbb{R},$$

trained with a numerical integration scheme is a *ContinuousNet* if it is convergent to a similar error as the error obtained by using

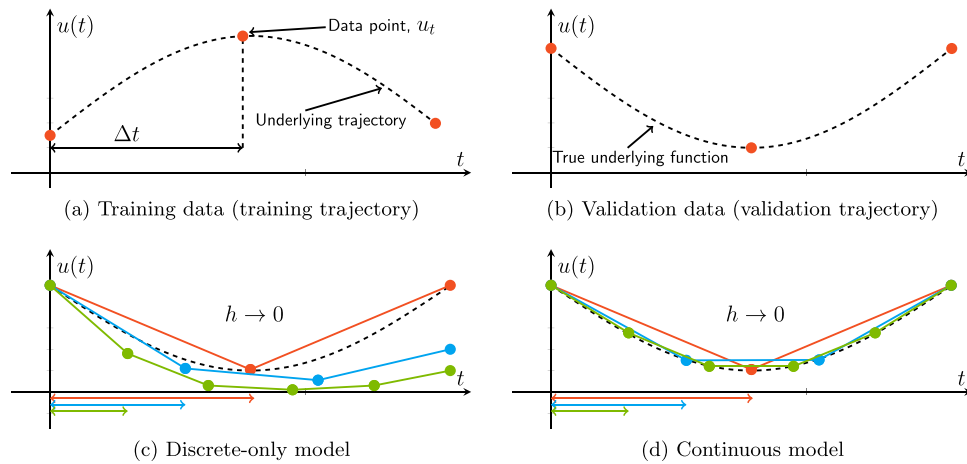


Fig. 1 Learning to predict discrete points versus learning continuous dynamics. **a** A NN model learns from a discrete set of points (red dots) that are generated by a dynamical system with an underlying continuous trajectory (dashed black line). **b** After training, the model should be able to predict future data points that are lying on the same or a different trajectory, including data points that are irregularly sampled. **c** A model that has only learned to predict the discrete data points might accurately fit future points on the same trajectory sampled at rate Δt (shown in red), but fail to predict points sampled at different rates. The blue and green lines evaluated with $h < \Delta t$ fall off of the underlying continuous trajectory. **d** A model that has learned an underlying continuous dynamics can converge to a continuous solution as $h \rightarrow 0$. In this case, the blue and green lines evaluated with $h < \Delta t$ get closer to the underlying continuous trajectory.

the original training time step,

$$\lim_{h \rightarrow 0} \text{Error}(h) \lesssim \text{Error}(\Delta t). \quad (4)$$

This convergence criteria is very similar to that of numerical analysis, whereby convergence is judged as $h \rightarrow 0$, and can be evaluated with a similar methodology adapted for the ML setting. The criteria of Eq. (4) represents a heuristic that takes into account the error from the learning process that can be observed in the error on the discrete-only validation task, $\text{Error}(\Delta t)$. (Note that we are not claiming that this method will guarantee that we have learned the true solution—that would require additional assumptions, well-known in scientific computing—simply that we have learned some underlying continuous model of the data.)

To illustrate the utility of our approach, we demonstrate how meaningfully continuous models that pass our convergence test enable both better interpolation and better extrapolation in multiple ways. We show that such models can resolve fine-scale features of the solution, despite being trained only on coarse data, including data that are irregularly spaced with non-uniform time intervals; can learn higher resolution solutions through learning continuous temporal dynamics from flow field snapshots; and can correctly predict trajectories starting at different initial conditions on which the model was not trained. We also demonstrate that our convergence test method is generally applicable to ML models. In addition, we derive theoretical error bounds for simple linear ODEs. Our results show promise in bridging between ML methodologies and scientific computing methodologies, by respecting both the fundamentals of ML and the fundamentals of science.

Results and discussion

In this section, we use the convergence test to demonstrate and identify discrete-overfitting of dynamics models.

We start by showing an example of our convergence test on a simple harmonic oscillator system. We then illustrate our convergence test on a variety of different scientific systems, demonstrating that our method can validate whether a trained dynamics model has learned (some) meaningfully continuous dynamics. Next, we show that models that pass this test can predict fine-scale solutions from coarsely spaced data. This includes:

predicting continuous temporal dynamics from flow fields; predicting trajectories starting at initial conditions on which the model was not trained; and predicting fine-scale solutions from coarse, irregularly spaced data. We then show that overfitting to the temporal discretization affects ML methods more generally than just with ODE-Nets.

Here, we only consider systems which are non-chaotic, non-divergent, and that are not extremely stiff, such that they can be handled by simple explicit Runge-Kutta integrators. More generally, learning the underlying “true” dynamics would require a test that involves a more sophisticated coupling of numerical and ML methodologies. This is not necessarily needed for many scientific ML tasks, including any of the improvements for predicting fine-scale solutions from coarsely spaced data that we discuss.

Example convergence method. We demonstrate our convergence test on a toy example. We sample discrete training data points from the linear differential equations describing the harmonic oscillator:

$$\frac{dx}{dt} = y; \quad \frac{dy}{dt} = -x. \quad (5)$$

We show the results in Fig. 2. Two ODE-Nets are trained on this data. Here, Fig. 2a–c use the forward Euler integration scheme, while Fig. 2d–f use the RK4 integration scheme. Both the Euler-Net (Fig. 2a) and RK4-Net (Fig. 2d) use the same linear network architecture $\mathcal{N}(x; \theta) = \theta x$ to approximate the ODE. In theory, a linear model can exactly represent the linear ODE; however, we will demonstrate that this does not happen. For the example in Fig. 2, the training data was generated from the analytical solution, spaced apart by the training timestep $\Delta t = 0.1$. To measure the performance as a continuous model at inference time, we integrate the model using a range of inference timesteps h . Figure 2b, e plot the results of the convergence test with the Euler-Net and the RK4-Net.

For the Euler-Net, the error when $h = \Delta t$ (the step size is equal to the temporal spacing in the training data) is very low, but it increases when h decreases. This is in contrast to the classical Euler numerical integration scheme, where the error decreases as h decreases. Thus, these results for Euler-Net do *not* pass the

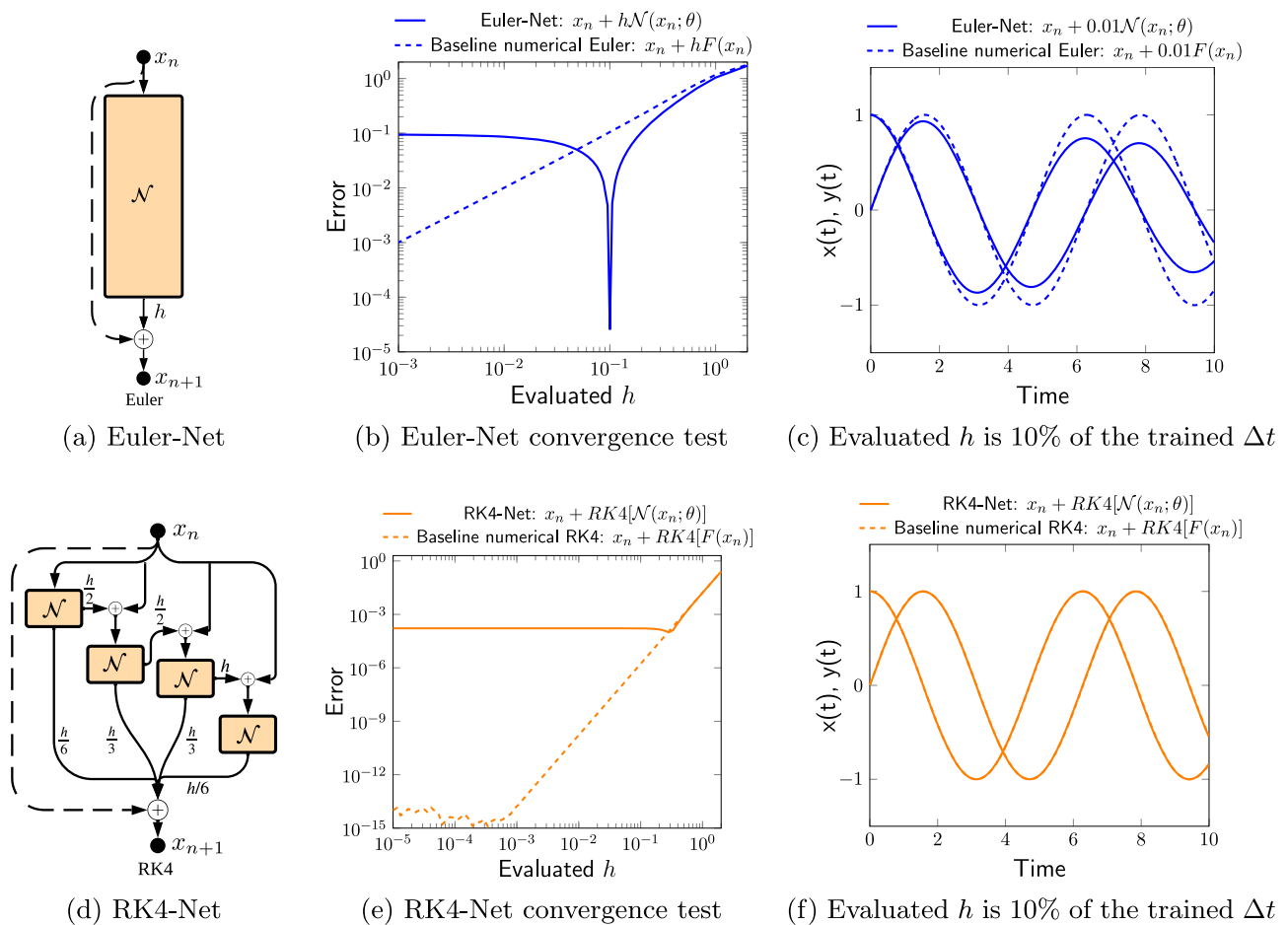


Fig. 2 Illustration of our convergence test with different ODE-Nets. **a** Schematic of an ODE-Net block, where the next timestep is obtained with the Euler method. In **(b, c)**, an Euler-Net is trained on discrete data points, spaced apart by some Δt (in this case, $\Delta t = 0.1$). **b** After training at one specific Δt , Euler-Net is evaluated at many different step sizes, h , both larger and smaller than Δt . The sharp dip at $h = 0.1$ demonstrates that the model achieves low error when $h = \Delta t$, i.e., it is a good discrete model (DiscreteNet) when the evaluated step size is the same as the Δt in the training data. However, when the model is evaluated at an h even slightly larger than or smaller than Δt , the error increases sharply; and thus the Euler-Net model does not pass the convergence test. This is in contrast to numerical integrators, where errors decrease monotonically as h decreases. **c** We visualize a trajectory where the Euler-Net is evaluated at a h one order of magnitude lower ($h = 0.01$) than the trained Δt . The resulting trajectory shows that the Euler-Net is unable to follow the baseline numerical Euler solution. **d** Schematic of an ODE-Net block, in which the RK4 numerical integration scheme is used to obtain the next timestep. As before, the RK4-Net is trained on discrete data points. This time, as **(e)** shows, the error converges monotonically to a fixed value as h decreases; and thus the RK4-Net model passes the convergence test, i.e., it is a good continuous model. The reason the RK4-Net error flattens and converges to a (non-zero but small) fixed value is due to ML-based error sources⁶⁴. The RK4 numerical integration scheme also converges to a (non-zero but very small) fixed value this time, due to numerical-based round-off errors⁶⁶. In **(f)**, the RK4-Net follows behavior similar to the RK4 numerical integration scheme when evaluated at a low h ($h < \Delta t$).

convergence test. In contrast, for the RK4-Net, the error decreases as $h \rightarrow 0$, and eventually it approaches and levels off at a fixed value. Notably, the error does not increase dramatically as it does with the Euler-Net. In this case, the RK4-Net has learned the right inductive biases to approximate an underlying continuous dynamics for the system.

We illustrate this further by showing an example trajectory at a specific evaluated h . In this case, both trained ODE-Nets are evaluated at $h = 0.01$ (a $10 \times$ increase in resolution in comparison to the training data) up to a final timestep. In Fig. 2c, the Euler-Net falls off of the true numerical Euler solution. It has clearly not learned the underlying continuous dynamics. In contrast, in Fig. 2f, the RK4-Net shows good correspondence with the true numerical RK4 solution.

Four prototypical dynamical systems. We consider canonical scientific dynamical systems: the non-linear pendulum, the Lotka-Volterra equations, the Cartesian pendulum, and the

double gyre fluid flow. The first two systems are non-linear dynamical systems; the Cartesian pendulum is a stiff dynamical system (which is difficult to solve with numerical methods without taking very small timesteps); and the double gyre fluid flow consists of vorticity fields describing a stream function. We provide more details about these dynamical systems in Supplementary Note 1: Details about considered dynamical systems. For each system, we sample data points from either the analytical solution or the numerical solution. The temporal spacing between the discrete data points is denoted as Δt , while the step size used to evaluate a trained ODE-Net is denoted as h .

Training setup. We train an ODE-Net with a numerical integration scheme (Euler or RK4) for each system. We use simple feed-forward networks with tanh activation functions. See Supplementary Note 2: Model architecture details for details on the architecture used. In every example, the exact same network architecture is used for both the Euler-Net and RK4-Net,

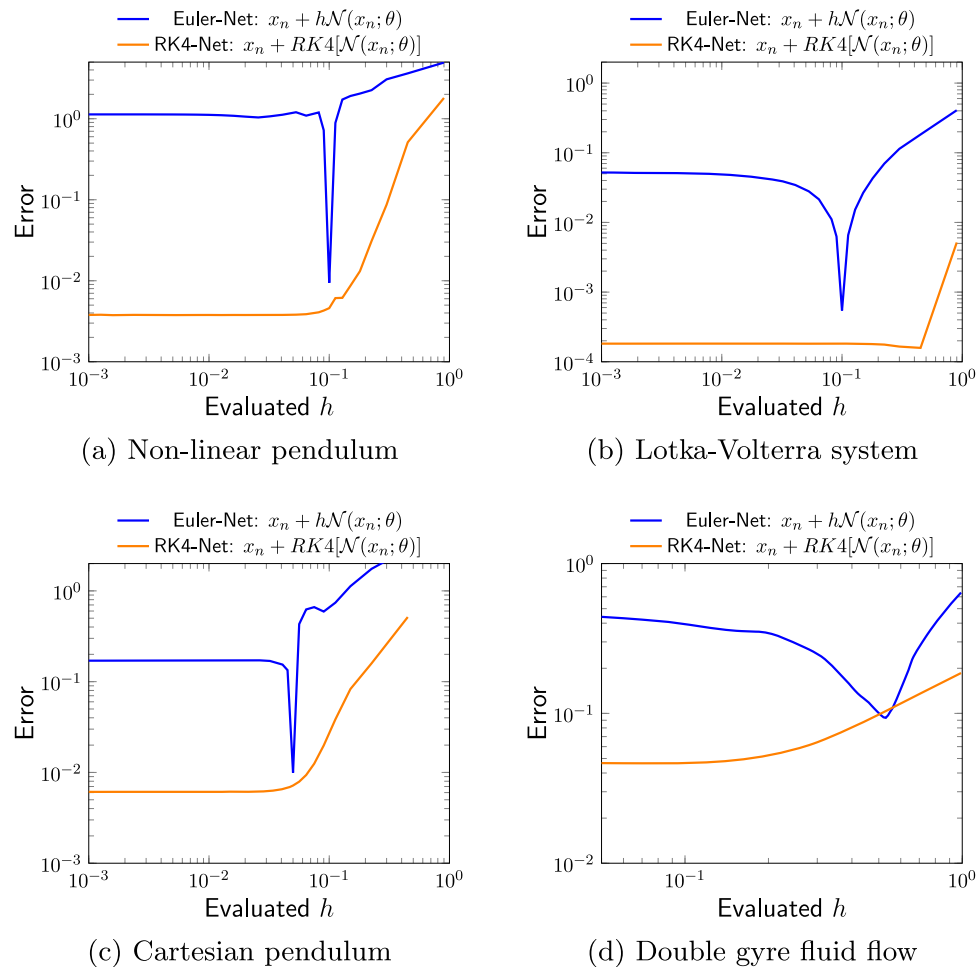


Fig. 3 Illustration of our convergence test on prototypical dynamical systems. We demonstrate the convergence test on multiple systems: (a) Non-linear pendulum (b) Lotka-Volterra system (c) Cartesian pendulum (d) Double gyre fluid flow. In each case, for the Euler-Net, the error does not monotonically decrease: error is low when $h = \Delta t$, but high at all other h evaluations. Thus, the Euler-Net does not pass the convergence test. In contrast, the RK4-Net error does monotonically decrease and converges to a fixed value, when the evaluated $h \rightarrow 0$. The RK4-Net does pass the convergence test.

respectively. We also include additional results for ODE-Nets trained on training data spaced apart by different Δt as well as an ODE-Net trained with the Midpoint numerical integration scheme, in Supplementary Note 3: Additional convergence test examples using ODE-Nets. For the double gyre fluid flow, we use a dynamic autoencoder architecture^{27, 29} to embed the high-dimensional input of flow field snapshots in some latent space. Specifically, we replace the linear discrete map in the architecture proposed by²⁷ with a linear ODE-block. This means that the model learns to predict the next timestep by integrating forward in latent space (using an Euler or RK4 numerical integration scheme) with step size $h = \Delta t$. Finally, the decoder translates the latent space vectors back to the flow field.

Results. The results of our method are shown in Fig. 3. In each case, the Euler-Net has low error when $h = \Delta t$ (i.e., evaluated at the same time spacing as the training data), but it has high error when evaluated at all other h , in particular smaller values of h . Thus, it does not pass the convergence test, and it has not learned a meaningfully continuous dynamics. It is a good discrete model, appropriate for data drawn from the same temporal discretization, but it has overfit to the temporal discretization. In contrast, the error during inference time of the RK4-Net steadily decreases when it is evaluated at lower h , eventually converging to a fixed basal level determined by the model and the noise properties of

the data. It has passed the convergence test, and it can be said to have learned a meaningfully-continuous model. We include additional convergence test results in Supplementary Figs. S1, S2, S3, S4.

Interpolation: predicting fine-scale solutions from coarse training data. Observational, discrete training data are limited in that they are measured at specific timesteps. To obtain a solution for the system in-between these timesteps, one must retake the data measurements again at finer timesteps. However, selecting a model that has learned meaningfully continuous dynamics should guarantee accurate evaluation at smaller timesteps, despite only training on coarse and/or irregularly spaced temporal data (i.e., measurements taken with large timesteps). By learning continuous dynamics, the trained ODE-Net model can be evaluated at any point in temporal space, and still yield a low error solution. In this case, one would not need to recollect training data with smaller Δt between data points; the learned ODE-Net can be used instead. Here, we demonstrate that fine-scale evaluation is possible by learning continuous temporal dynamics from flow fields for the double gyre flow example.

Results. We consider two models: the Euler-Net which did not pass the convergence test, and the RK4-Net which did pass the

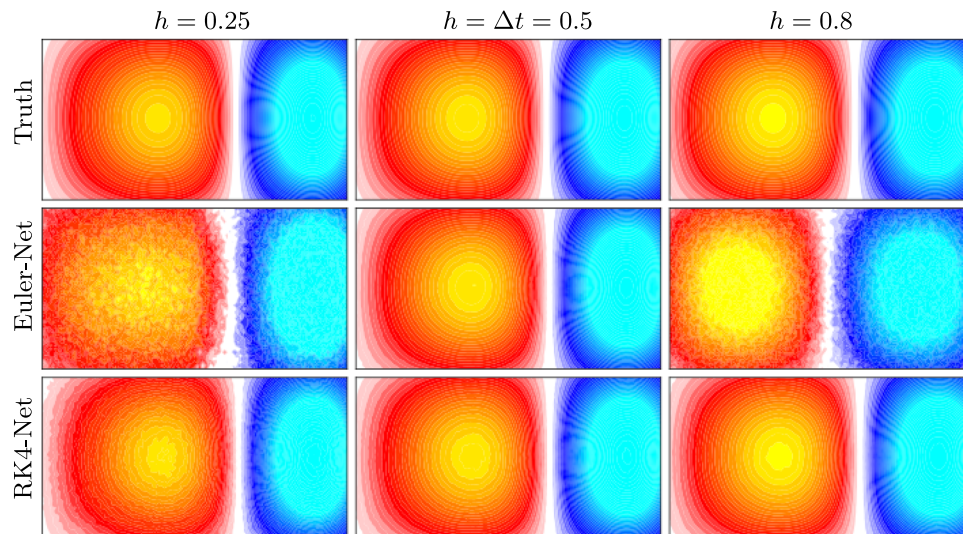


Fig. 4 Double gyre fluid flow: reconstructing fine-scale flow fields from coarse training data. The training data for this problem consists of vorticity field snapshots of the dynamical system taken at $\Delta t = 0.5$. In the images above, the red region is rotating in one direction, and the blue region is rotating in the opposite direction. After training an Euler-Net and an RK4-Net, both models are evaluated at different h (both when $h > \Delta t$ and $h < \Delta t$) at a final timestep. We show the evaluation results for this final timestep, T , at $h = 0.25$, $h = 0.5 = \Delta t$, and $h = 0.8$. The Euler-Net (which fails the convergence test) approximates a solution close to the true solution at a timestep in the training data but does poorly at the other timesteps; it does not capture the fluid flow behavior and gives a grainier solution. The RK4-Net (which passes the convergence test) is able to output solutions that have close correspondence to the true solution; it successfully interpolates the fine-scale flow fields that are in-between the training data snapshots, resulting in a much higher frame rate solution.

convergence test (see Fig. 3). In Fig. 4, we show the flow field snapshots that result from both models being evaluated at different timesteps. The Euler-Net is only able to approximate the true solution at the training data timestep (in this case, $h = \Delta t = 0.5$). It cannot match the true solution at the other timesteps, and it gives a poor approximation that does not capture the flow behavior. In contrast, the RK4-Net has good correspondence to the true solution even when it is evaluated at timesteps that were not in the training data. Thus, our convergence test method has allowed us to choose a model that can recover fine-scale solutions of the system, while only having access to coarse-scale measurements during training.

Extrapolation: predicting trajectories for new initial conditions. For a given system, temporal trajectories start at some initial condition. Measurements are taken for one trajectory at one initial condition, and then must be taken separately for other trajectories with different initial conditions. Selecting a model that has learned a meaningfully continuous dynamics circumvents this: after training a model on data points sampled from one (or more) trajectories, the model should be able to extrapolate and predict accurate solutions for new initial conditions.

Training setup. We look at the non-linear pendulum (described in Eq. 27 in Supplementary Note 1: Details about considered dynamical systems). Here, θ is the initial condition representing the position of the pendulum in time. The phase portrait of this system (representing the true solution trajectories), showing $\frac{d\theta}{dt}$ against θ , is shown in Fig. 5a. An Euler-Net and an RK4-Net are trained on trajectories, spaced apart by $\Delta t = 0.1$, starting at certain initial conditions (shown by the black lines in Fig. 5b). We then pick a test set of a number of different initial conditions that were not in the training data. The Euler-Net and RK4-Net start at these test initial conditions and are both evaluated at a finer $h = 0.001$, representing a $100 \times$ increase in resolution. Note that we saw in Fig. 3 that the Euler-Net did not pass the convergence

test (i.e., it had high error when evaluated at $h \ll \Delta t$), while the RK4-Net did pass the test.

Results. The results of predicting trajectories starting at different test initial conditions are shown in Fig. 5. The Euler-Net is unable to predict these trajectories and quickly falls off of the phase plot lines corresponding to the true solution. In contrast, the RK4-Net is able to predict the trajectories, starting at different test initial conditions with good correspondence to the true solution. Thus, we see that it is critical to find a model that passes the convergence test and is able to learn a continuous dynamics to succeed at this extrapolation task.

Irregularly sampled training data. It is typically the case that scientific data collection includes measurements that are taken with some amount of imprecision. For example, the measurement of interest is not always taken at the exact same Δt every time, due to issues such as jitter in the measurement device. Measurements may also be skipped: for example, a measurement is only available at $t = \Delta t$ and $t = 3\Delta t$ because the measurement at $t = 2\Delta t$ was lost or skipped. Thus, reconstructing the correct trajectory when the measurements are non-uniformly spaced is important in numerous science and engineering problems. Here, we look at an example of using the convergence test to correctly select a meaningfully continuous model in the case of the non-linear pendulum with irregularly spaced temporal data with non-uniform temporal intervals.

Training setup. An example distribution of irregularly sampled training data is shown in Fig. 6a. The baseline Δt is 0.05, subject to jitter and frameskipping errors. An Euler-Net and an RK4-Net are both trained on these temporal data points, where the values of Δt are input into the integration schemes. (That is, at every given measurement, the timestep jitter was also recorded to use in training.) Both ODE-Nets are then evaluated at a very low h (approximately $100 \times$ lower than the general distribution of the

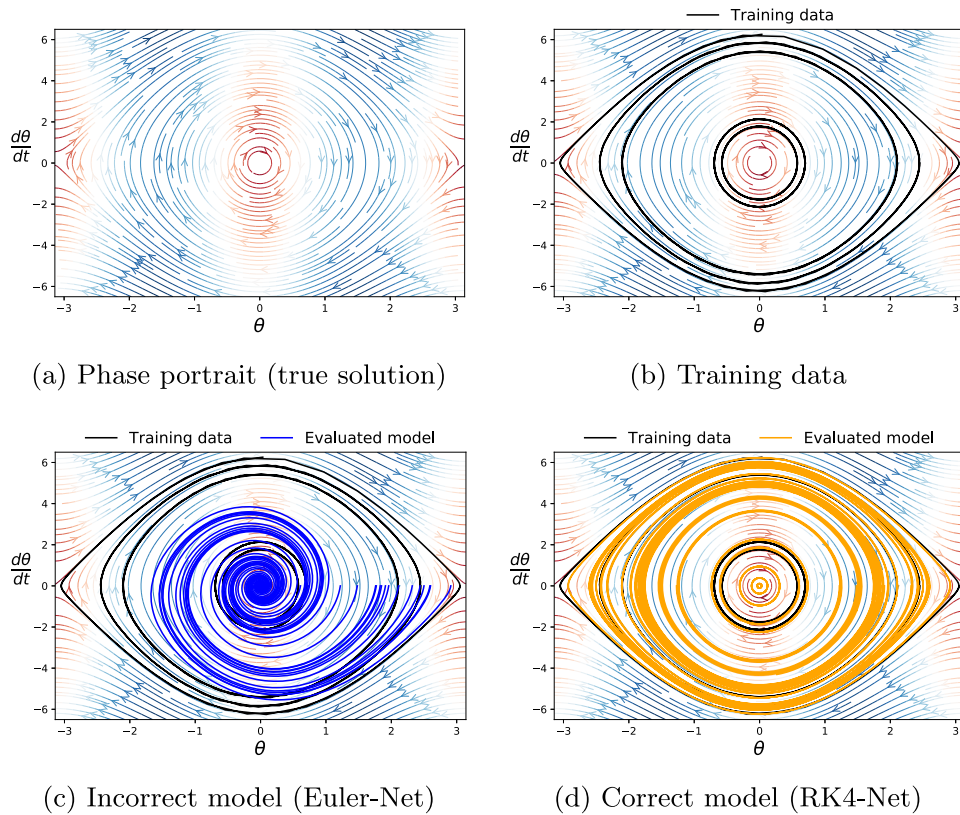


Fig. 5 Non-linear pendulum: extrapolation to predict initial condition trajectories on which the model was not trained. **a** Phase portrait of true solution. **b** ODE-Net models are trained on randomly chosen initial conditions (different θ values), which have temporal points spaced apart by $\Delta t = 0.1$. Each model is then evaluated on a different set of initial conditions (not in the training data) at much smaller step sizes ($h = 0.001$, $100 \times$ higher resolution). **c** After evaluation, the Euler-Net quickly falls off of the phase plot lines corresponding to the true solution for the different test trajectories. **d** The RK4-Net, which has learned a continuous dynamics, is able to extrapolate to different trajectories (starting at different initial conditions), with good correspondence to the phase plot lines of the true solution.

training data points) to generate a time series plot. Note, that we run the convergence test on both ODE-Nets.

Results. The Euler-Net quickly falls off of the continuous solution (Fig. 6b). Conversely, the RK4-Net follows the continuous solution with good accuracy, including at timesteps not in the training data (Fig. 6c). Thus, it is clear that the RK4-Net has learned a meaningfully continuous dynamics while the Euler-Net has not. This is confirmed by RK4-Net passing the convergence test, but Euler-Net not passing it (Fig. 6d). The dip for Euler-Net appears at the average Δt in the training data, which is slightly larger than 0.05 due to the measurement noise.

Sparse identification of nonlinear dynamical systems. Here, we demonstrate that overfitting to the temporal discretization affects ML methods (in the context of dynamical systems) more generally. To illustrate this, we consider the SINDy learning approach, which is a class of methods for system identification⁵.

The SINDy method uses the following model structure to represent the dynamics,

$$\frac{dx}{dt} = \Xi \phi(x), \quad (6)$$

where Ξ is a matrix of learnable parameters and $\phi(x)$ is a set of non-linear basis functions which correspond to potential terms in the underlying system. A linear optimizer is used to fit the parameters Ξ to the data, with a sparsity constraint. The sparsity constraint identifies the subset of relevant basis elements in $\phi(x)$ to reveal an interpretable dynamics model.

In most real applications, the time derivatives, dx_n/dt , cannot be measured directly and instead need to be approximated from the observations x_n . The common approach in SINDy is to use finite differences to approximate dx_n/dt from the data^{47, 48}. (This treatment of time derivatives, where SINDy differentiates the data, is in contrast to the ODE-Net method, which integrates the model.) The finite difference operator $FD(x_{n\dots})$ is applied over the dataset as a pre-processing step. This yields the following set of N discrete equations, which is optimized for Ξ over all observations n :

$$FD(x_{n\dots}) = \Xi \phi(x_n), \quad (7)$$

where $FD(x_{n\dots})$ is a finite difference approximation using the region of points around time index n . Using the series of N equations, Ξ is learned using specialized algorithms designed to seek sparsity, such as LASSO regularization or sequential threshold least squares⁵. (This is again in contrast with the ODE-Net method, which uses gradient descent nonlinear optimization.) The discretization order of the finite difference pre-processing is a hyperparameter of SINDy. The first order accurate finite difference (here referred to as FD-1) results in a point-wise approximation of,

$$\frac{x_{n+1} - x_n}{\Delta t} = \Xi \phi(x_n). \quad (8)$$

Note that when rearranged, this is equivalent to the forward Euler integrator:

$$x_{n+1} = x_n + \Delta t \Xi \phi(x_n). \quad (9)$$

Higher-order finite difference stencils can also be used to increase the accuracy of the time derivative approximation. The second-

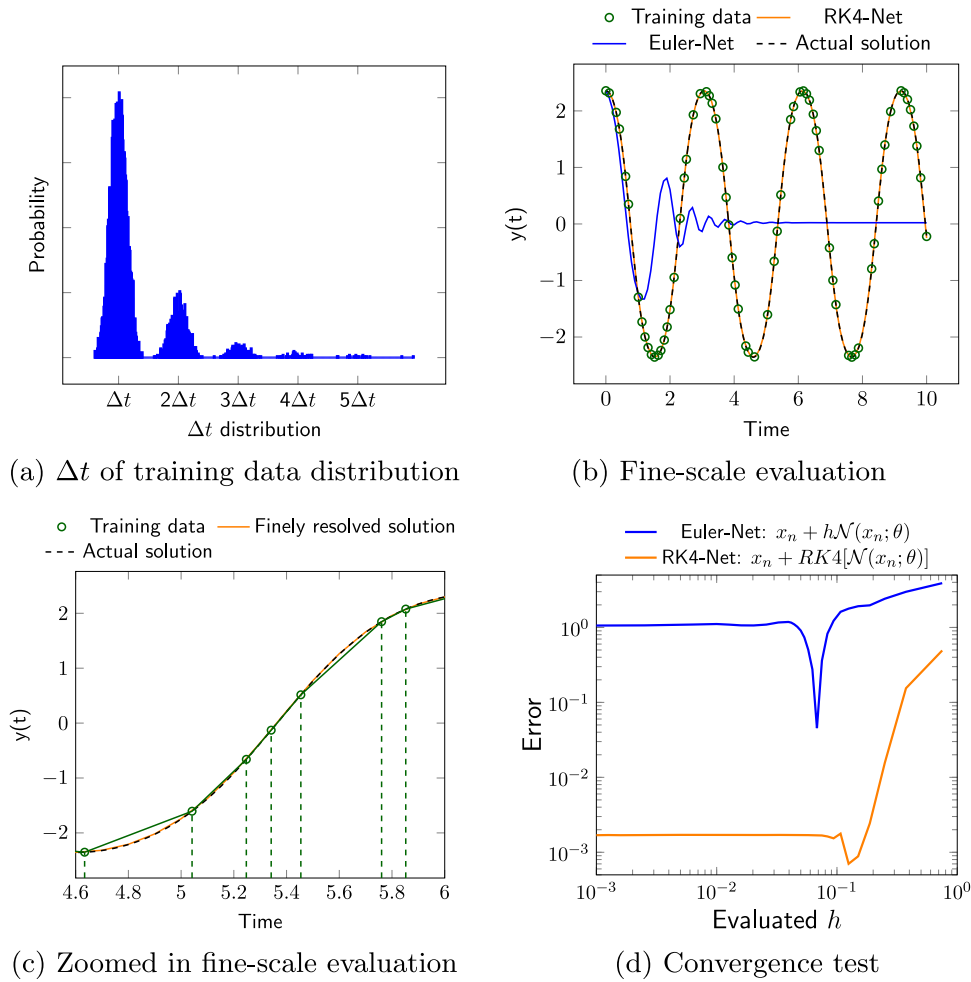


Fig. 6 Learning continuous dynamics from irregularly spaced discrete points. **a** Training data distribution for a scientific problem where temporal data measurements are taken with some amount of imprecision (e.g., the measurement of interest is not always taken at the exact same Δt) and/or measurements are skipped (e.g., we only have a measurement at Δt and $3\Delta t$ because the measurement at $2\Delta t$ failed or was lost). **b** Different ODE-Nets are trained on the irregularly spaced data (indicated by green dots), as denoted by the green circles. Then, the trained models are evaluated at a very low h (where $h \ll \Delta t$). The RK4-Net (orange) is able to learn a continuous dynamics and follow the continuous solution over time (indicated by the dashed line), while the Euler-Net (blue) does not. **c** The RK4-Net (orange) is able to reconstruct the fine-scale, high resolution solution (indicated by dashed line), with good correspondence to the continuous solution, from the coarse, irregularly spaced training data (indicated by green dots). **d** The RK4-Net (orange) passes the convergence test, but the Euler-Net (blue) does not.

order stencil (FD-2) (analogous, but not equivalent, to Midpoint) can be expressed as

$$\frac{1}{2\Delta t}(x_{n+1} - x_{n-1}) = \Xi\phi(x_n). \quad (10)$$

and the fourth-order stencil (FD-4) (analogous, but not equivalent, to RK4) can be expressed as,

$$\frac{1}{\Delta t} \left(-\frac{1}{12}x_{n+2} + \frac{2}{3}x_{n+1} - \frac{2}{3}x_{n-1} + \frac{1}{12}x_{n-2} \right) = \Xi\phi(x_n). \quad (11)$$

Training Setup. We look at the harmonic oscillator described in Eq. (5), and the non-linear pendulum (described in Eq. 27 in Supplementary Note 1: Details about considered dynamical systems). We use the PySINDy implementation^{47, 48} to train models on these trajectories. We train three different SINDy models on the trajectories, altering the finite difference (FD) approximation order of accuracy: FD-1 is a first-order two-point stencil (analogous to Euler-Net), FD-2 is a second-order three-point stencil (analogous to Midpoint-Net), and FD-4 is a fourth-order five-point stencil (analogous to RK4-Net). The SINDy

model is plugged into the convergence test as F , such that the same Runge-Kutta integrators are used for trajectory prediction. For the harmonic oscillator, the training data is spaced apart by $\Delta t = 0.1$, while for the non-linear pendulum, we look at examples where the training data is spaced apart by $\Delta t = 0.05$ and $\Delta t = 0.1$.

Results. We run our convergence test on the different SINDy models. The results of our method are shown in Fig. 7. In each case, the FD-1 model has low error when $h = \Delta t$ (i.e., evaluated at the same time spacing as the training data), but it has high error when evaluated at all other h , and especially smaller values of h . Thus, it does not pass the convergence test, and it has not learned a meaningfully continuous dynamics. The FD-1 model shows a sharp dip because it is overfit to forward Euler. In this case, the stencil and integrator correspond to the exact same algebraic structure. Similar to when the models were trained via ODE-Nets, we see that FD-1 has overfit to the temporal discretization. In contrast, the error during the inference time of the FD-4 model steadily decreases when it is evaluated at lower h , eventually converging to a fixed basal level. It has passed the convergence test, and it has learned a meaningfully-continuous model.

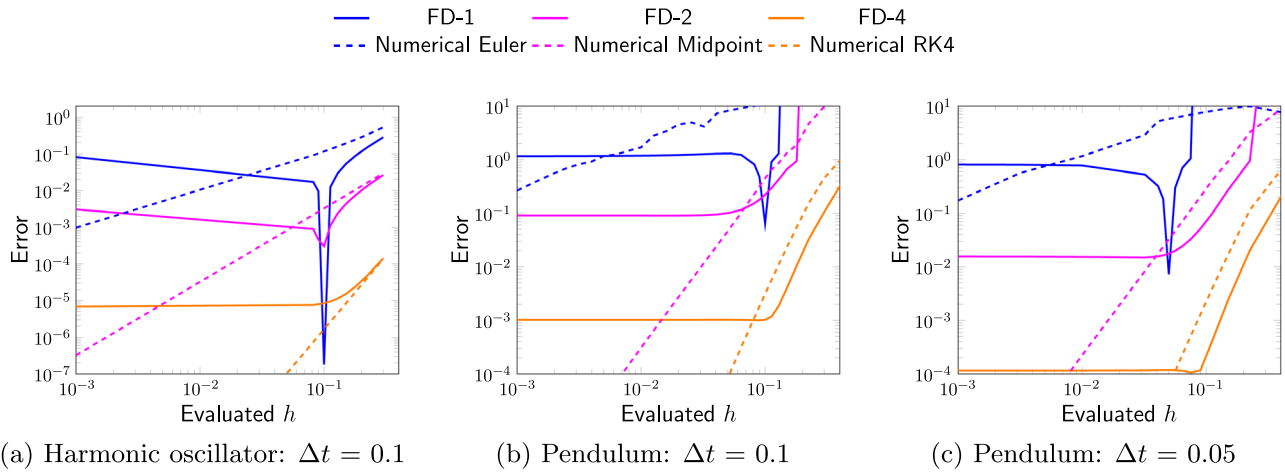


Fig. 7 SINDy method: additional convergence tests. Instead of training from dynamical data with an ODE-Net model, SINDy is used instead to learn a sparse polynomial basis. The accuracy of the time derivative approximation is altered by using a different order of central finite difference stencils. Here, FD-1 is the first-order two-point stencil, FD-2 is the second-order three-point stencil, and FD-4 is the fourth-order five-point stencil. In this example, FD-1 is analogous to Euler-Net, FD-2 is analogous to Midpoint-Net, and FD-4 is analogous to RK4-Net. In **(a)**, the convergence test method is shown on the harmonic oscillator, where the convergence test is replicated by training via the SINDy method. In **(b, c)**, the convergence test is applied when training on the non-linear pendulum via the SINDy method. We observe similar results as when training using an ODE-Net: the lower-order approximation (FD-1) overfits to the Δt in the training data, failing the convergence test; and the higher-order approximation (FD-4) passes the convergence test.

Conclusion

One of the great challenges in scientific ML is to learn continuous dynamics for physical systems—either “the” underlying continuous dynamics, or “a” continuous dynamics that leads to good predictive results for the spatial/temporal regime of interest for the ML model—such that the learned ML model can be trusted to give accurate and reliable results. ML models are trained on discrete points, and typical ML training/testing methodologies are not aware of the continuity properties of the underlying problem from which the data are generated. Here, we have developed a methodology, and we showed that convergence (an important criteria used in numerical analysis) can be used for selecting models that have a strong inductive bias towards learning meaningfully continuous dynamics. Standard ODE-Net approaches, as well as common SINDy methods, both popular in recent years within the ML community, often do not pass this convergence test. In contrast, models that pass this convergence test have favorable properties. For instance, models that learned underlying continuous dynamics can be evaluated at lower or higher resolutions. Our results suggest that principled numerical analysis methods can be coupled with existing ML training/testing methodologies to deliver upon the promise of scientific ML more generally.

Many more concrete directions are of course raised by our methodology. One direction has to do with developing analogous tests to be used for less well-posed dynamical systems. Such systems are of interest in scientific ML, and such tests will be of greatest interest when one needs to obtain “the” correct underlying continuous solution (e.g., to identify correctly qualitative long-term behavior⁴⁶), rather than “a” continuous solution, which is often sufficient for ML prediction tasks. Another direction has to do with whether we can develop analogous tests appropriate for adaptive time-stepping methods, symplectic integrators, and other commonly used numerical simulation methods such as those for optimal control problems using NNs and associated Hamilton-Jacobi partial differential equations^{49, 50}. Work subsequent to the posting of the initial technical report version of this paper has addressed the continuous-discrete equivalence question for learning operators^{51, 52}, and likely our methodology provides a way to operationalize that in practice. A

final direction has to do with whether one can obtain strong theoretical results, e.g., ML-style generalization bounds, to guide the use of methods such as these. Recent theoretical and empirical results suggest that this will be challenging^{53–57}, at least when using traditional approaches to ML-style generalization bounds. Our success in combining principled numerical analysis methods with existing ML methodologies also leads one to wonder whether we can use a posteriori error bound analysis methods to develop practically useful a posteriori generalization bounds for problems such as those we have considered.

Methods

The basic problem of numerical analysis is to solve problems from continuous mathematics using a discrete computer. The area has a rich history for describing the consistency and convergence behavior of numerical methods for approximating continuous functions^{58, 59}. Here, we expand on the methods we used in Results and Discussion.

Criteria of Classical Numerical Analysis. Given an initial value $x(0) = x_0$, we can discretize Eq. (2) along the node points $t_n = n\Delta t$ for $n = 0, 1, \dots, N$ by evaluating the following integral equation:

$$x_{n+1} = x_n + \int_{t_n}^{t_n + \Delta t} F(x(s)) ds, \quad (12)$$

where $x_n = x(t_n)$, and Δt is the discrete timestep. Typically, we are not able to compute an analytic solution for the integral, and thus we rely on numerical schemes to approximate $\int_{t_n}^{t_n + \Delta t} F(x(s)) ds$.

There are many different types of numerical integration schemes to approximate the integral in Eq. (12). These have different trade-offs between computational efficiency and accuracy. One such scheme, the forward Euler discretization, can be written as:

$$x_{n+1} = x_n + \Delta t F(x_n). \quad (13)$$

This is a first-order one-step method, where the global error (the error over all of the timesteps) is proportional to the step size, i.e., $\mathcal{O}(\Delta t)$, meaning that the error gets smaller as Δt decreases. There are also higher-order integration schemes. One popular higher-

order scheme is the Runge-Kutta 4 (RK4) discretization, which takes the following form:

$$\begin{aligned}
 i_1 &= F(x_n) \\
 i_2 &= F\left(x_n + \frac{\Delta t}{2} \cdot i_1\right) \\
 i_3 &= F\left(x_n + \frac{\Delta t}{2} \cdot i_2\right) \\
 i_4 &= F(x_n + \Delta t \cdot i_3) \\
 x_{n+1} &= x_n + \frac{1}{6} \Delta t (i_1 + 2i_2 + 2i_3 + i_4).
 \end{aligned}
 \tag{14}$$

Here, the global error is proportional to the step size to the fourth power, i.e., $O(\Delta t^4)$; and thus as Δt gets smaller, the error gets smaller much more quickly than with the forward Euler scheme. In general, the global error can be written as $O(\Delta t^p)$, where p denotes the order of accuracy.

Classical numerical integration typically starts by assuming that there exists a true underlying continuous-time system, which is then replaced by a discrete-time problem whose solution approximates that of the continuous problem. However, discretizing the problem introduces an error, and concepts such as stability, convergence, and consistency can be used to quantify the error of the discrete solution^{60–62}.

In the following, we describe the error bounds in the traditional scientific computing context where the system dynamics are known exactly, in which case the only approximation error comes from numerical integration in time. We specifically focus on numerical convergence because this will give us a mechanism to analyze ML models. However, note that stability and consistency are also of interest⁵⁹. Let $x(t_n)$ denote the true solution of a dynamical system of interest; and let $\bar{x}_n^{\Delta t}$ denote a numerical solution after n steps with step size Δt . We use N to denote the maximum number of time steps such that $T = t_N = N\Delta t$ is the final time. Decreasing Δt requires increasing N (the number steps taken to arrive at T), and vice versa. Then, $x(T) = x(t_N)$ is the true solution at the final time, while $\bar{x}_N^{\Delta t}$ is the numerical solution at the final time. Convergence quantifies the global error (the cumulative error of all iterations) of a numerical algorithm.

Definition 2. (Convergent Numerical Approximation). A numerical one-step method for solving $\frac{dx(t)}{dt} = F(x(t))$, with initial condition $x(0) = x_0$, is said to be convergent if and only if the error tends to zero as Δt goes to zero:

$$\lim_{\Delta t \rightarrow 0} \|x(t_N) - \bar{x}_N^{\Delta t}\|_2 = 0.
 \tag{15}$$

Of course, in numerical practice, the error does not converge to zero. Instead, it levels off at some base level determined typically by the level of numerical precision used to describe the data, as observed in 2e.

The specific metric for quantifying the approximation error across the sequence is somewhat arbitrary. Moreover, there is the problem that the numerical method can potentially converge to the wrong solution⁶³. Thus, to ensure that a numerical method is not only convergent but also consistent, one can use the mean error,

$$\lim_{\Delta t \rightarrow 0} \frac{1}{N} \sum_{n=0}^N \|x(t_n) - \bar{x}_n^{\Delta t}\|_2 = 0,$$

or the maximum error across all N points in time,

$$\lim_{\Delta t \rightarrow 0} \max_{n=1,2,\dots,N} \|x(t_n) - \bar{x}_n^{\Delta t}\|_2 = 0.$$

If, as the step size Δt decreases, the largest absolute error between the numerical solution $\bar{x}_n^{\Delta t}$ and the exact solution $x(t_n)$ also

decreases, then the numerical approximation converges towards the solution of the continuous system. In the limit of $\Delta t \rightarrow 0$, the numerical solution converges to the exact solution and the error converges to zero, or to some base level determined by machine precision and numerical round-off noise.

This convergence criteria is also a test for continuity in the solution: as $\Delta t \rightarrow 0$, the time interval between adjacent numerical solutions (e.g., at t_n, x_n , and at t_{n+1}, x_{n+1}) also decreases towards zero. Thus, the numerical solution collapses onto a continuous solution as $\Delta t \rightarrow 0$.

Validation of a new integration method involves multiple stages. Consistency, convergence, and stability can be theoretically proven for a rather small class of ODEs (typically only linear ODEs). Thus, the method will be evaluated empirically with a real implementation on a problem of interest. In practice, a convergence test is used, where the numerical integration scheme is used to predict trajectories for a range of Δt and compared to an analytical solution or to an overrefined solution. The errors are verified to approach zero at the correct rate, at least until they flatten out at some base level. The combination of theoretical proof of consistency, stability, and convergence on simple systems such as linear ODEs, combined with the empirical demonstration of convergence on ODEs of interest, is typically viewed as sufficient to vet the method. Empirical convergence tests are a standard integration test method for scientific programs, e.g., they are regularly run to automatically catch bugs.

A convergence test for ODE-nets. We now describe a convergence test, based on the discussed convergence criteria, to validate properties of an ODE-Net solution. The fact that ODE-Nets are embedded in a numerical integration scheme enables us to use convergence analysis methods that are well-known for studying classical numerical analysis problems. To start, we know that the numerical integrator itself will be convergent if it is given the true f , but we do not know if the ODE-Net will be convergent when an approximate ML model \mathcal{N} is used to approximate f . The convergence test is used to determine whether an ODE-Net has learned a meaningfully continuous model for the underlying problem of interest.

Suppose that we are given an ODE-Net \mathcal{N} that is trained with a numerical integration scheme (such as Euler or RK4) from t_n to t_{n+1} with stepsize Δt :

$$\begin{aligned}
 \bar{x}_{n+1} &= \text{ODESolve}[\mathcal{N}(x_n; \theta), \text{start} = t_n, \text{end} = t_{n+1}, \\
 &\quad \text{step} = \Delta t, \text{scheme}].
 \end{aligned}
 \tag{16}$$

Following Definition 2, we compute the global error of the ODE-Net \mathcal{N} as it approaches some fixed value b as the time step h goes to zero:

$$\lim_{h \rightarrow 0} \|x_N - \bar{x}_N^h\|_2 = b.
 \tag{17}$$

Here too, in the ML setting, the error does not necessarily converge to zero. This is analogous to the classical numerical analysis setting, where the numerical analysis test typically converges to a non-zero value determined by the numerical round-off error (e.g., see the floor in Fig. 2e). Unlike in the classical numerical analysis setting, even in the absence of numerical errors the b of an ML model will be greater than zero. For an ODE-Net, the numerical value of b depends on the model architecture, integration method, the optimizer, and the noise properties of the data⁶⁴. The value of b will elucidate the convergence properties of the trained ODE-Net.

Computing an error metric in the ML setting requires additional consideration because we do not necessarily have access to the underlying exact solution at arbitrary points in time. Instead, we are restricted to the information that is provided by a given validation

set of discrete data points $\mathcal{T} = \{x_0, x_1, x_2, \dots, x_N\}$, where each point is spaced apart by the Δt between observations. A naive metric to compute the global error in this setting is simply to consider the 2-norm between the end point of the validation trajectory and the predicted value:

$$\text{Error}(h) = \|x_N - \bar{x}_N^h\|_2. \quad (18)$$

However, this metric is susceptible to noise and edge cases. Computing the error over all points \mathcal{T} is difficult using (19) because the inferred trajectory has a different number of points than the validation trajectory. To mitigate this issue, we suggest to compute the global error on a subset of points \mathcal{S} from the validation/test trajectories, which is a set of indexes into the original dataset \mathcal{T} ; e.g., $\mathcal{S} = \{0, 9, 18, \dots\}$ for every 9 points spaced by $9\Delta t$. This allows for inferring the trajectory of h computing the error over the subset as follows:

$$\text{Error}(h) = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \|x_n - \bar{x}_k^h\|_2, \quad (19)$$

where k is the index into the inferred trajectory corresponding to the index into the validation trajectory n such that \bar{x}_k^h is the point that lines up at the same time as x_n . Note that we can only use certain timesteps h during inference because the solution points must align perfectly with those in the subset trajectory.

Given this setup, we use the term ContinuousNet to refer to an ODE-Net model that also exhibits these convergence properties, as per Definition 1. To evaluate this property, we can apply the same convergence test procedure used in traditional numerical analysis and scientific computing, but with the modifications necessary for it work on training data. Further, it is necessary to apply a weaker heuristic to judge convergence because there will be residual optimization error at $\text{Error}(\Delta t)$, as per Eq. (4). The procedure is as follows.

Given the learned ODE-Net, we first infer a validation/test trajectory on the original stepsize Δt and evaluate the global error, $\text{Error}(\Delta t)$. This is the standard procedure for evaluating a discrete model, and this error value informs its accuracy at inferring discrete points in a sequence. Then, to further evaluate whether the model is convergent and continuous, we consider a range of h values which are both smaller and larger than the step size Δt used during training. For example, if the ODE-Net was trained on $\Delta t = 0.1$, we evaluate the ODE-Net on the validation/test trajectory for $h \in [10^{-3}, \dots, 10^1]$. Specifically, for a given set of inference timesteps $\{h_1, h_2, \dots, h_p\}$, our proposed convergence test iterates over the elements h_i , and executes the following two steps:

1. Evaluate the pre-trained ODE-Net using Eq. (16) on the time interval $[t_0, t_T]$, using step size h_i .
2. Calculate the error between the ODE-Net solution $\bar{x}_n^{h_i}$ and points in the test data, $\text{Error}(h_i)$.

Algorithms 1 and 2 in Supplementary Note 4: Algorithm for the Convergence Test summarize this procedure. As with many other numerical convergence tests, our proposed algorithm is subject to a heuristic threshold. In practice, we observe that the difference between a model that passes our test condition and one that does not is pronounced. A non-continuous model results in an error that is orders of magnitude larger, as compared to a continuous model, when h is taken smaller than Δt of the training data. In other words, our convergence test performs a form of model selection by selecting for models that learn inductive biases towards meaningfully continuous dynamics. In the following, we will demonstrate why convergence is an important consideration for ML model selection. We also discuss the two other criterion of classical numerical analysis, consistency and stability, in Supplementary Note 4: Algorithm for the Convergence Test.

In practice, we can also evaluate our convergence test with different starting points x_0 and the respective final time step, x_N , and then average the error across the different runs. We highly recommend this to ensure that the same behavior occurs irrespective of start and end point.

Error analysis in an idealized learning setting. We provide a theoretical framework for the convergence test, analyzing the discretization error of one-step numerical integration schemes in an idealized setting. For additional details, see Supplementary Note 5: Theoretical Derivations. Specifically, we consider the problem of learning the simple scalar linear ODE,

$$\frac{dx(t)}{dt} = \lambda x(t), \quad (20)$$

where $\lambda \in \mathbb{R}$ denotes a scalar parameter, and $x \in \mathbb{R}$ denotes the state at a given point in time. It is well known that the function $x(t) = e^{\lambda t} x_0$ is a solution of this system, given the initial condition $x(t) = x_0 \in \mathbb{R}^{65}$. In the following, we assume a similar setting as before: we are given a set of discrete data points $\mathcal{D} = \{x_0, x_1, x_2, \dots, x_N\}$ produced by the linear system and spaced by Δt . Our aim is to learn a scalar ODE-Net model,

$$\frac{dx(t)}{dt} = w x(t), \quad (21)$$

parameterized by the learnable weight parameter w . Following the ODE-Net process, we discretize the model with a numerical integration scheme and then optimize the squared error loss,

$$\min_w \sum_{x \in \mathcal{D}} (x_{n+1} - \text{ODESolve}[w x_n, \text{step} = \Delta t, \text{scheme}])^2. \quad (22)$$

We assume that the data are noise-free and can therefore be represented by its analytical solution, $x_{n+1} = e^{\lambda \Delta t} x_n$. When the loss is optimized, the time-discretization step introduces its own unique source of error into the learning process, one that is independent of noise, numerical error, or optimization error. The error stems from the fact that any one-step consistent numerical integration scheme, when applied to a linear ODE, will result in a truncated Taylor series expansion with p terms, where p is the accuracy order of the scheme⁵⁹. Thus, the ML model cannot recover the exact parameters of the underlying ODE. The following lemma makes this issue explicit.

Lemma 1. In absence of any other optimization errors, a scalar ODE-Net can at best obtain a weight parameter w by minimizing Eq. (22) that, for certain timesteps and integrators, satisfies the following polynomial equation,

$$\exists p, \Delta t, w \text{ s.t. } \sum_{i=0}^p \frac{\Delta t^i}{i!} w^i = e^{\lambda \Delta t}. \quad (23)$$

The proof is given in Supplementary Note 6: Optimization of w does not learn λ . For finite Δt , this equation clearly satisfies $w \neq \lambda$ if $p \ll \infty$. There are situations where this equation can be solved for values of w that will set the loss in Eq. (22) to zero for all possible data points x_n . In the limit as $\Delta t \rightarrow 0$, there is always a solution at $w = \lambda$ for any p . Moreover, for practical settings there is at least one root when p is odd for any Δt ; when $p = 2$ (RK2) and $\Delta t \leq \log(2)/|\lambda|$; or when $p = 4$ (RK4) and $\Delta t < 1.307/|\lambda|$. This equation can be used to find analytical expressions for w for simple integrators; see Supplementary Note 7: Example evaluations of the analytical equations.

From this result, we can characterize the difference between the ML model and the target ODE. In addition, in practice, it is (almost always) only possible to learn a perturbed version, \tilde{w} , of w due to noise, limited numerical precision, and optimization errors. Let ϵ denote an additive perturbation away from the the

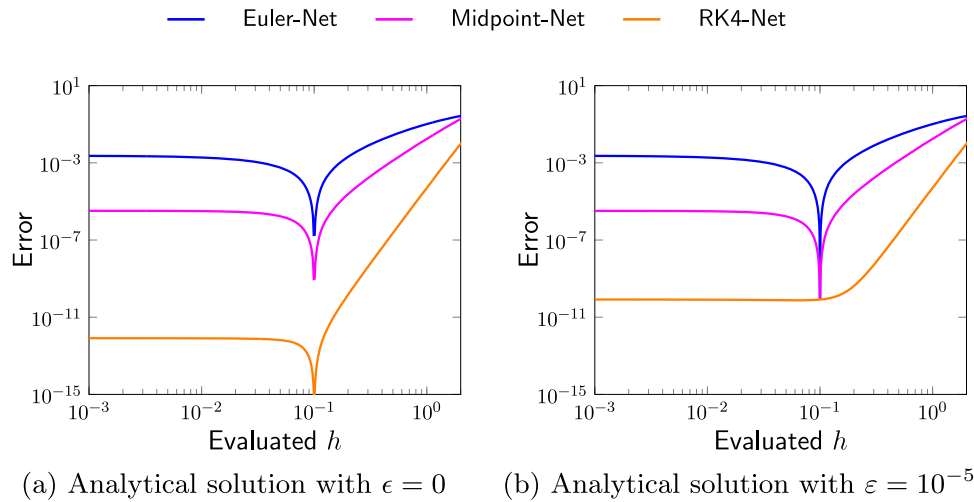


Fig. 8 Illustration of our convergence test for linear ODEs. The analytically derived global error estimate, $Error(h)$, using the ideally learned linear ODE. Our results replicate the empirically performed convergence tests. **a** All methods achieve zero global error at $h = \Delta t$, but converge to finite values as $h \rightarrow 0$. **b** We introduce the ϵ error parameter. midpoint and RK4 achieve a finite error of comparable magnitudes at $h = \Delta t$. Overfitting is not observed for RK4 with $\epsilon = 10^5$ because the $|\lambda - w|$ error is less significant than $|\epsilon|$.

optimum of the minimization problem for an observed model due to these sources of errors, $\tilde{w} = w + \epsilon$. The following theorem bounds the error between the ML and ODE model, due to the overfitting of Eq. (23) in presence of additive error sources.

Theorem 1. If an optimal weight parameter can be found by Eq. (23), the approximation error introduced by scalar ODE-Net is bounded by $|w - \lambda| \leq c \Delta t^p$, where c is a constant proportional to the Lipschitz continuity constants of λx and $w x$. In the presence of additive numerical error ϵ , the bound is,

$$|\tilde{w} - \lambda| \leq |w - \lambda| + |\epsilon| \leq c \Delta t^p + |\epsilon|. \quad (24)$$

The proof is given in Supplementary Note 8: Approximation errors introduced by scalar ODE-Nets. This bound shows that the user needs to increase the order of accuracy of the training scheme p in order to reduce the error between the learned parameter and the true ODE parameter.

In practice, we can only measure $Error(h)$ using a set of data points. Using the above results, we can analyze the expected behavior of the convergence test by bounding the global error using Eq. (24). Figure 8 plots the global error bound given concrete values of λ , Δt , and ϵ . As can be seen, the theoretically derived global error for the scalar ODE exhibits the same behavior as the empirical applications of the convergence test. We can use the global error to further derive expected bounds on the the key points of the convergence test.

Corollary 1. When a scalar ODE-Net is evaluated with the timestep that was used for training, the leading term of the global error is proportional to the optimization error ϵ for a $k \propto T|x_0|$:

$$Error(\Delta t) = k|\epsilon| + \mathcal{O}(\Delta t|w||\epsilon| + \Delta t|\epsilon|^2). \quad (25)$$

The proof is given in Supplementary Note 9: Observable quantities and the convergence test. The $c \Delta t^p$ error term in Eq. (24) between w and λ is cancelled out, and the observed value is smaller than $|\tilde{w} - \lambda|$. Therefore, the global error only observes the difference between w and \tilde{w} , resulting from the model optimization error. Note that this value can become very small as the optimization error decreases. By applying the convergence test, we are able to extract an estimate of $|\tilde{w} - \lambda|$, as given in the following.

Corollary 2. In the limit of decreasing the timestep size during inference, the global error approaches a constant factor (b) based on the bound in Eq. (24). It approaches at a rate of h^q , where q is the accuracy order of the ODE integration scheme used at inference time:

$$\lim_{h \rightarrow 0} Error(h) = |\tilde{w} - \lambda| + \mathcal{O}(h^q) \leq k|\epsilon| + kc\Delta t^p + \mathcal{O}(h^q). \quad (26)$$

The proof is given in Supplementary Note 9: Observable quantities and the convergence test. Given these bounds, we can see how using Eq. (4) as a threshold yields $(b - Error(\Delta t)) \propto c \Delta t^p$. Therefore, the comparison between b and $Error(\Delta t)$ allows for the quantitative estimation for the magnitude of the term $c \Delta t^p$, which describes the error that is induced by the numerical discretization scheme used for training.

In summary, our analysis shows that there are two types of errors in the process of learning the dynamics of a linear scalar ODE using ODE-Nets. These errors can be measured using the data by evaluating $Error(\Delta t)$ and $\lim_{h \rightarrow 0} Error(h)$. This illustrates the power of our proposed convergence criterion Eq. (4). Moreover, even if ϵ is small, it is required to increase the order of accuracy of the training scheme in order to further decrease the error between \tilde{w} and λ .

Data availability

The source code for the data generation will be made available at <https://github.com/a1k12/learning-continuous-physics->.

Code availability

The code will be made available at <https://github.com/a1k12/learning-continuous-physics->.

Received: 18 July 2022; Accepted: 16 October 2023;

Published online: 03 November 2023

References

1. Robinson, R. C. *An introduction to dynamical systems: continuous and discrete*, vol. 19 (American Mathematical Soc., 2012).

2. Brunton, S. L. & Kutz, J. N. *Data-driven science and engineering: Machine learning, dynamical systems, and control* (Cambridge University Press, 2019).
3. Calinon, S., Li, Z., Alizadeh, T., Tsagarakis, N. G. & Caldwell, D. G. Statistical dynamical systems for skills acquisition in humanoid robots. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 323–329 (IEEE, 2012).
4. Peters, J. R. *Machine learning of motor skills for robotics* (University of Southern California, 2007).
5. Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl Acad. Sci.* **113**, 3932–3937 (2016).
6. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236* (2018).
7. Keller, R. T. & Du, Q. Discovery of dynamics using linear multistep methods. *SIAM J. Numer. Anal.* **59**, 429–455 (2021).
8. Rudy, S., Alla, A., Brunton, S. L. & Kutz, J. N. Data-driven identification of parametric partial differential equations. *SIAM J. Appl. Dynamical Syst.* **18**, 643–660 (2019).
9. Jin, P., Zhang, Z., Zhu, A., Tang, Y. & Karniadakis, G. E. SympNets: Intrinsic structure-preserving symplectic networks for identifying hamiltonian systems. *Neural Netw.* **132**, 166–179 (2020).
10. Lutter, M., Ritter, C. & Peters, J. Deep Lagrangian Networks: Using physics as model prior for deep learning. *International Conference on Learning Representations* (2019).
11. Chen, Z., Zhang, J., Arjovsky, M. & Bottou, L. Symplectic recurrent neural networks. *International Conference on Learning Representations* (2019).
12. Erichson, N. B., Azencot, O., Queiruga, A., Hodgkinson, L. & Mahoney, M. W. Lipschitz recurrent neural networks. *International Conference on Learning Representations* (2020).
13. Rusch, T. K., Mishra, S., Erichson, N. B. & Mahoney, M. W. Long expressive memory for sequence modeling. *arXiv preprint arXiv:2110.04744* (2021).
14. Wang, R., Maddix, D., Faloutsos, C., Wang, Y. & Yu, R. Bridging physics-based and data-driven modeling for learning dynamical systems. In *Learning for Dynamics and Control*, 385–398 (PMLR, 2021).
15. Lim, S. H., Erichson, N. B., Hodgkinson, L. & Mahoney, M. W. Noisy recurrent neural networks. *Adv. Neural Inform. Processing Sys.* **34**, 5124–5137 (2021).
16. Jiahao, T. Z., Hsieh, M. A. & Forgoston, E. Knowledge-based learning of nonlinear dynamics and chaos. *Chaos: Interdiscip. J. Nonlinear Sci.* **31**, 111101 (2021).
17. Négiar, G., Mahoney, M. W. & Krishnapriyan, A. Learning differentiable solvers for systems with hard constraints. In *The Eleventh International Conference on Learning Representations* <https://openreview.net/forum?id=vdv6CmGksr0> (2023).
18. Morton, J., Witherden, F. D. & Kochenderfer, M. J. Deep variational Koopman models: Inferring Koopman observations for uncertainty-aware dynamics modeling and control. *arXiv preprint arXiv:1902.09742* (2019).
19. Lambert, N., Amos, B., Yadan, O. & Calandra, R. Objective mismatch in model-based reinforcement learning. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, vol. 120 of *Proc. Machine Learn. Res.* 761–770 (PMLR, 2020).
20. Li, Y., He, H., Wu, J., Katabi, D. & Torralba, A. Learning compositional Koopman operators for model-based control. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=H1ldzA4tPr> (2020).
21. Bachnas, A., Tóth, R., Ludlage, J. & Mesbah, A. A review on data-driven linear parameter-varying modeling approaches: A high-purity distillation column case study. *J. Process Control* **24**, 272–285 (2014).
22. Karniadakis, G. E. et al. Physics-informed machine learning. *Nat. Rev. Phys.* **3**, 422–440 (2021).
23. Manojlović, I. et al. Applications of Koopman mode analysis to neural networks. *arXiv preprint arXiv:2006.11765* (2020).
24. Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R. & Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. *Adv. Neural Inform. Process. Sys.* **34** (2021).
25. Bar-Sinai, Y., Hoyer, S., Hickey, J. & Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proc. Natl Acad. Sci.* **116**, 15344–15349 (2019).
26. Pestourie, R., Mroueh, Y., Rackauckas, C., Das, P. & Johnson, S. G. Physics-enhanced deep surrogates for PDEs. *arXiv preprint arXiv:2111.05841* (2021).
27. Erichson, N. B., Muehlebach, M. & Mahoney, M. W. Physics-informed autoencoders for Lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866* (2019).
28. Otto, S. E. & Rowley, C. W. Linearly recurrent autoencoder networks for learning dynamics. *SIAM J. Appl. Dynamical Syst.* **18**, 558–593 (2019).
29. Azencot, O., Erichson, N. B., Lin, V. & Mahoney, M. W. Forecasting sequential data using consistent Koopman autoencoders. *International Conference on Machine Learning* 475–485 (2020).
30. Dubois, P., Gomez, T., Planckaert, L. & Perret, L. Data-driven predictions of the Lorenz system. *Phys. D: Nonlinear Phenom.* **408**, 132495 (2020).
31. Asadi, K., Misra, D., Kim, S. & Littman, M. L. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320* (2019).
32. Chen, R. T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* **31** (2018).
33. Ruthotto, L. & Haber, E. Deep neural networks motivated by partial differential equations. *J. Math. Imaging Vis.* **62**, 352–364 (2020).
34. Queiruga, A., Erichson, N. B., Hodgkinson, L. & Mahoney, M. W. Stateful ODE-Nets using basis function expansions. *Adv. Neural Inf. Process. Syst.* **34**, 21770–21781 (2021).
35. Massaroli, S., Poli, M., Park, J., Yamashita, A. & Asama, H. Dissecting neural ODEs. *Adv. Neural Inf. Process. Syst.* **33**, 3952–3963 (2020).
36. Zhang, T. et al. ANODEV2: A coupled neural ODE framework. *Adv. Neural Inf. Process. Syst.* **32**, 5151–5161 (2019).
37. Weinan, E. A proposal on machine learning via dynamical systems. *Commun. Math. Stat.* **5**, 1–11 (2017).
38. Rubanova, Y., Chen, R. T. & Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. *Adv. Neural Inf. Process. Syst.* **32**, 5320–5330 (2019).
39. Greydanus, S. J., Dzumba, M. & Yosinski, J. Hamiltonian neural networks. *Adv. Neural Inf. Process. Syst.* **32** (2019).
40. Du, J., Futoma, J. & Doshi-Velez, F. Model-based reinforcement learning for semi-markov decision processes with neural ODEs. *Adv. Neural Inf. Process. Syst.* **33**, 19805–19816 (2020).
41. Greydanus, S., Lee, S. & Fern, A. Piecewise-constant neural ODEs. *arXiv preprint arXiv:2106.06621* (2021).
42. Chen, R. T., Amos, B. & Nickel, M. Learning neural event functions for ordinary differential equations. *International Conference on Learning Representations* (2021).
43. Jia, J. & Benson, A. R. Neural jump stochastic differential equations. *Adv. Neural Inf. Process. Syst.* **32**, 9847–9858 (2019).
44. Queiruga, A. F., Erichson, N. B., Taylor, D. & Mahoney, M. W. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389* (2020).
45. Ott, K., Katiyar, P., Hennig, P. & Tiemann, M. ResNet after all: Neural ODEs and their numerical solution. *International Conference on Learning Representations* (2021).
46. Rico-Martinez, R., Krischer, K., Kevrekidis, I. G., Kube, M. C. & Hudson, J. L. Discrete- vs. continuous-time nonlinear signal processing of Cu electro-dissolution data. *Chem. Eng. Commun.* **118**, 25–48 (1992).
47. de Silva, B. et al. Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *J. Open Source Softw.* **5**, 2104 (2020).
48. Kaptanoglu, A. A. et al. Pysindy: A comprehensive python package for robust sparse system identification. *J. Open Source Softw.* **7**, 3994 (2022).
49. Nakamura-Zimmerer, T., Gong, Q. & Kang, W. QRnet: Optimal regulator design with LQR-augmented neural networks. *IEEE Control Syst. Lett.* **5**, 1303–1308 (2021).
50. Darbon, J., Langlois, G. P. & Meng, T. Overcoming the curse of dimensionality for some Hamilton-Jacobi partial differential equations via neural network architectures. *Res. Math. Sci.* **7**, 1–50 (2020).
51. Bartolucci, F. et al. Are neural operators really neural operators? frame theory meets operator learning. Tech. Rep. Preprint: arXiv:2305.19913 (2023).
52. Raonic, B. et al. Convolutional neural operators for robust and accurate learning of PDEs. Tech. Rep. Preprint: arXiv:2302.01178 (2023).
53. Martin, C. H. & Mahoney, M. W. Traditional and heavy-tailed self regularization in neural network models. In *Proceedings of the 36th International Conference on Machine Learning*, 4284–4293 (2019).
54. Martin, C. H. & Mahoney, M. W. Heavy-tailed Universality predicts trends in test accuracies for very large pre-trained deep neural networks. In *Proceedings of the 20th SIAM International Conference on Data Mining* (2020).
55. Martin, C. H., Peng, T. S. & Mahoney, M. W. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nat. Commun.* **12**, 1–13 (2021).
56. Martin, C. H. & Mahoney, M. W. Post-mortem on a deep learning contest: a simpson's paradox and the complementary roles of scale metrics versus shape metrics. *arXiv preprint arXiv:2106.00734* (2021).
57. Hodgkinson, L., Simsekli, U., Khanna, R. & Mahoney, M. W. Generalization bounds using lower tail exponents in stochastic optimizers. *International Conference on Machine Learning* (2022).
58. Moín, P. *Fundamentals of engineering numerical analysis* (Cambridge University Press, 2010).
59. LeVeque, R. J. & LeVeque, R. J. *Numerical methods for conservation laws*, vol. 132 (Springer, 1992).
60. Dahlquist, G. Convergence and stability in the numerical integration of ordinary differential equations. *Mathematica Scandinavica* 33–53 (1956).
61. Arnold, D. N. Stability, consistency, and convergence of numerical discretizations. *Encyclopedia of Applied and Computational Mathematics* 1358–1364 (2015).

62. Kirby, R. M. & Silva, C. T. The need for verifiable visualization. *IEEE Computer Graph. Appl.* **28**, 78–83 (2008).
63. Thompson, D. B. Numerical methods 101-convergence of numerical models. *USGS Staff–Published Research* 115 (1992).
64. Bottou, L. & Bousquet, O. The tradeoffs of large scale learning. *Adv. Neural Inf. Process. Syst.* **20** (2008).
65. Hirsch, M. W., Smale, S. & Devaney, R. L. *Differential equations, dynamical systems, and an introduction to chaos* (Academic press, 2012).
66. Chaitin-Chatelin, F. & Frayssé, V. *Lectures on finite precision computations* (SIAM, 1996).

Acknowledgements

We would like to thank Annan Yu and Krishna Harsha Reddy Kothapalli for valuable discussions and feedback. Moreover, we would like to thank all the reviewers for their helpful and constructive feedback. A.S.K. was supported by Laboratory Directed Research and Development (LDRD) funding under Contract Number DE-AC02-05CH11231 at LBNL and the Alvarez Fellowship in the Computational Research Division at LBNL. M.W.M. would like to acknowledge the DOE, NSF, and ONR for providing partial support of this work. N.B.E. would like to acknowledge support from NSF (DMS-2319621), DOE (AC02-05CH11231), and NERSC (DE-AC02-05CH11231). Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

Author contributions

A.S.K. and A.F.Q. contributed equally. A.S.K., A.F.Q., N.B.E., and M.W.M. all contributed to the manuscript discussion.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s42005-023-01433-4>.

Correspondence and requests for materials should be addressed to Aditi S. Krishnapriyan.

Peer review information *Communications Physics* thanks Ljupco Todorovski and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Reprints and permission information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

This is a U.S. Government work and not under copyright protection in the US; foreign copyright protection may apply 2023