FINDE: NEURAL DIFFERENTIAL EQUATIONS FOR FINDING AND PRESERVING INVARIANT QUANTITIES

Takashi Matsubara
Osaka University
Toyonaka, Osaka, 560–8531 Japan
matsubara@sys.es.osaka-u.ac.jp

Takaharu Yaguchi
Kobe University
Kobe, Hyogo, 657–8501 Japan
yaguchi@pearl.kobe-u.ac.jp

ABSTRACT

Many real-world dynamical systems are associated with first integrals (a.k.a. invariant quantities), which are quantities that remain unchanged over time. The discovery and understanding of first integrals are fundamental and important topics both in the natural sciences and in industrial applications. First integrals arise from the conservation laws of system energy, momentum, and mass, and from constraints on states; these are typically related to specific geometric structures of the governing equations. Existing neural networks designed to ensure such first integrals have shown excellent accuracy in modeling from data. However, these models incorporate the underlying structures, and in most situations where neural networks learn unknown systems, these structures are also unknown. This limitation needs to be overcome for scientific discovery and modeling of unknown systems. To this end, we propose *first integral-preserving neural differential equation (FINDE)*. By leveraging the projection method and the discrete gradient method, FINDE finds and preserves first integrals from data, even in the absence of prior knowledge about underlying structures. Experimental results demonstrate that FINDE can predict future states of target systems much longer and find various quantities consistent with well-known first integrals in a unified manner.

1 INTRODUCTION

Modeling and predicting real-world systems are fundamental aspects of understanding the world in natural science and improving computer simulations in industry. Target systems include chemical dynamics for discovering new drugs (Raff et al., 2012), climate dynamics for climate change prediction and weather forecasting (Rasp et al., 2020; Trigo & Palutikof, 1999), and physical dynamics of vehicles and robots for optimal control (Nelles, 2001). In addition to image processing and natural language processing (Devlin et al., 2018; He et al., 2016), neural networks have been actively studied for modeling dynamical systems (Nelles, 2001). Their history dates back to at least the 1990s (see Chen et al. (1990); Clouse et al. (1997); Levin & Narendra (1995); Narendra & Parthasarathy (1990); Sjöberg et al. (1994); Wang & Lin (1998) for examples). Recently, two notable but distinct families have been proposed. Physics-informed neural networks (PINNs) directly solve partial differential equations (PDEs) given as symbolic equations (Raissi et al., 2019). Neural ordinary differential equations (NODEs) learn ordinary differential equations (ODEs) from observed data and solve them using numerical integrators (Chen et al., 2018). Our focus this time is on NODEs.

Most real-world systems are associated with *first integrals* (a.k.a. invariant quantities), which are quantities that remain unchanged over time (Hairer et al., 2006). First integrals arise from intrinsic geometric structures of systems and are sometimes more important than superficial dynamics in understanding systems (see Appendix A for details). Many previous studies have extended NODEs by incorporating prior knowledge about first integrals and attempted to accurately learn a target system. Greydanus et al. (2019) proposed the Hamiltonian neural network (HNN), which employs a neural network to approximate Hamilton's equation, thereby conserving the system energy called the Hamiltonian. Finzi et al. (2020a) proposed neural network architectures that conserve linear and angular momenta by utilizing the graph structure. Finzi et al. (2020b) also extended an HNN to a system with holonomic constraints, which led to first integrals such as a pendulum length.

Table 1: Comparison of Related Studies on Preservation of First Integrals.

	Energy	Momentum	Mass	Constraint	Learning invariants	Exact conservation
NODE (Chen et al., 2018)						
HNN (Greydanus et al., 2019)	✓					
LieConv (Finzi et al., 2020a)	✓	✓				
DGNet (Matsubara et al., 2020)	✓		✓			✓
CHNN (Finzi et al., 2020b)	✓			✓		
NPM (Yang et al., 2020)				✓	✓	✓
Continuous FINDE (proposed)	✓	✓	✓	✓	✓	
Discrete FINDE (proposed)	✓	✓	✓	✓	✓	✓

Matsubara et al. (2020) proposed a model that preserves the total mass of a discretized PDE. These studies have demonstrated that the more prior knowledge a neural network has about first integrals, the more accurate their dynamics prediction. See Table 1 for comparisons.

Previous studies have mainly attempted to preserve known first integrals for better computer simulations. However, in situations where a neural network learns a target system, it is naturally expected that first integrals associated with the target system are unknown, and it is not clear which of the above methods are available. Therefore, this study proposes *first integral-preserving neural differential equation* (FINDE) to find and preserve unknown first integrals from data in a unified manner. FINDE has two versions for continuous and discrete time; these have the following advantages.

Finding First Integrals Many studies have designed architectures or operations of neural networks to model continuous-time dynamics with known types of first integrals. However, the underlying geometric structures of a target system are generally unknown in practice. In contrast, FINDE finds various types of first integrals from data in a unified manner and preserves them in predictions. For example, from an energy-dissipating system, FINDE can find first integrals other than energy. FINDE can find not only known first integrals, but also unknown ones. Hence, FINDE can lead to scientific discoveries.

Combination with Known First Integrals FINDE can be combined with previously proposed neural networks designed to preserve known first integrals, such as HNNs. In addition, when some first integrals are known in advance, they can also be incorporated into FINDE to avoid rediscovery. Therefore, FINDE is available in various situations.

Exact Preservation of First Integrals The first integral associated with a continuous-time system is destroyed after the dynamics is temporally discretized for computer simulations. By leveraging the discrete gradient, the discrete-time version of FINDE preserves first integrals exactly (up to rounding errors) in discrete time and further improves the prediction performance.

2 Background and Related Work

First Integrals Let us consider a time-invariant differential system $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ on an N-dimensional manifold $\mathcal{M}$, where $\boldsymbol{u}$ denotes the system state and $f : \mathcal{M} \to \mathcal{T}_{\boldsymbol{u}}\mathcal{M}$ represents a vector field on $\mathcal{M}$. For simplicity, we suppose the manifold $\mathcal{M}$ to be a Euclidean space $\mathbb{R}^N$.

Definition 1 (first integral)**.** *A quantity $V : \mathcal{M} \to \mathbb{R}$ is referred to as a first integral of a system $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ if it remains constant along with any solution $\boldsymbol{u}(t)$, i.e., $\frac{\mathrm{d}}{\mathrm{d}t}V(\boldsymbol{u}) = 0$.*

If a differential system $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ has K functionally independent first integrals $V_1, \dots, V_K$, the solution $\boldsymbol{u}(t)$ given an initial value $\boldsymbol{u}_0$ stays at the $(N-K)$-dimensional submanifold

$$\mathcal{M}' = \{\boldsymbol{u} \in \mathcal{M} : V_1(\boldsymbol{u}) = V_1(\boldsymbol{u}_0), \dots, V_K(\boldsymbol{u}) = V_K(\boldsymbol{u}_0)\}. \tag{1}$$

The tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}' \subset \mathcal{T}_{\boldsymbol{u}}\mathcal{M}$ of the submanifold $\mathcal{M}' \subset \mathcal{M}$ at a point $\boldsymbol{u}$ is the orthogonal complement to the space spanned by the gradients $\nabla V_k(\boldsymbol{u})$ of the first integrals V_k for $k = 1, \dots, K$;

$$\mathcal{T}_{\boldsymbol{u}}\mathcal{M}' = \{\boldsymbol{w} \in \mathcal{T}_{\boldsymbol{u}}\mathcal{M} : \nabla V_k(\boldsymbol{u})^\top \boldsymbol{w} = 0 \text{ for } k = 1, \dots, K\}. \tag{2}$$

Conversely, if the time-derivative f at point $\boldsymbol{u}$ is on the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$ for certain functions V_k's, the quantities V_k's are first integrals of the system $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$; it holds that $\frac{\mathrm{d}}{\mathrm{d}t}V_k(\boldsymbol{u}) = \nabla V_k(\boldsymbol{u})^\top \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = \nabla V_k(\boldsymbol{u})^\top f(\boldsymbol{u}) = 0$.

One of the most well-known first integrals is the Hamiltonian H, which represents the system energy of a Hamiltonian system. Noether's theorem states that a continuous symmetry of a system leads to a conservation law (and hence a first integral) (Hairer et al., 2006). A Hamiltonian system is symmetric to translation in time, and the corresponding first integral is the Hamiltonian. Symmetries to translation and rotation in space lead to the conservation of linear and angular momenta. However, not all first integrals are related to symmetries. A pendulum can be expressed in Cartesian coordinates, and then the rod length constrains the mass position. This type of constraint is called a holonomic constraint and leads to first integrals. Models of disease spreads and chemical reactions have the total mass (population) as the first integral. Also for a system described by a PDE, the total mass is sometimes a first integral (Furihata & Matsuo, 2010). See Appendix A for the classes of dynamics, their geometric structures, and related studies to find or preserve first integrals.

First Integrals in Numerical Analysis For computer simulations, differential systems are discretized in time and solved by numerical integration, causing numerical errors (which is composed of temporal discretization errors and rounding errors). Moreover, the geometric structures of the system are often destroyed, and the corresponding first integrals are no longer preserved. A common remedy is a symplectic integrator, which preserves the symplectic structure and accurately integrates Hamiltonian systems (Hairer et al., 2006). However, the Ge–Marsden theorem states that a symplectic integrator only approximately conserves the Hamiltonian (Zhong & Marsden, 1988). Hence, many numerical schemes have also been investigated to preserve first integrals exactly, while these schemes cannot preserve the symplectic structure. Some examples are shown below.

Let the superscript $s \in \{0, 1, \ldots, S\}$ denote the state $\boldsymbol{u}^s$ or time t^s at the s-th time step, and $\Delta t^s = t^{s+1} - t^s$ denote a time-step size. A projection method uses a numerical integrator to predict the next state $\tilde{\boldsymbol{u}}^{s+1}$ from the current state $\boldsymbol{u}^s$ and then projects the state $\tilde{\boldsymbol{u}}^{s+1}$ onto the submanifold $\mathcal{M}'$ (Gear, 1986; Hairer et al., 2006, Section IV.4). The projected state $\boldsymbol{u}^{s+1}$ preserves the first integrals V_k. In particular, the projected state $\boldsymbol{u}^{s+1}$ is obtained by solving the optimization problem

$$\arg\min_{\boldsymbol{u}^{s+1}} \|\boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1}\| \text{ subject to } V_k(\boldsymbol{u}^{s+1}) - V_k(\boldsymbol{u}^s) = 0 \text{ for } k = 1, \ldots, K. \tag{3}$$

The local coordinate method defines a coordinate system on the neighborhood of the current state $\boldsymbol{u}^s$ and integrates a differential equation on it (Potra & Yen, 1991; Hairer et al., 2006, Section IV.5). The discrete gradient method defines a discrete analogue to a differential system and integrates it in discrete time, thereby preserving the Hamiltonian exactly (up to rounding errors) in discrete time (Furihata & Matsuo, 2010; Gonzalez, 1996; Hong et al., 2011).

Neural Networks to Preserve First Integrals NODE defines the right-hand side f of a differential system $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ using a neural network in the most general way with no associated first integrals (Chen et al., 2018). NODE is a universal approximator to ODEs and can approximate any ODE with arbitrary accuracy if there is an infinite amount of training data (Teshima et al., 2020). In practice, the amount of training data is limited, and prior knowledge about the target system is helpful for learning (see Sannai et al. (2021) for the case with convolutional neural networks (CNNs)). HNN (Greydanus et al., 2019) assumes the target system to be a Hamiltonian system in the canonical form, thereby guaranteeing various properties of Hamiltonian systems by definition, including the conservation of energy and preservation of the symplectic structure in continuous time (Hairer et al., 2006). Some studies have employed a symplectic integrator for HNN to preserve the energy and symplectic structure with smaller numerical errors (Chen et al., 2020). LieConv and EMLP-HNN employ neural network architectures with translational and rotational symmetries to preserve momenta (Finzi et al., 2020a; 2021). CHNN incorporates a known holonomic constraint in the dynamics (Finzi et al., 2020b). Deep conservation extracts latent dynamics of a PDE system and preserves a quantity of interest by forcing its flux to be zero (Lee & Carlberg, 2021). HNN++ also guarantees the conservation of mass in PDE systems by using a coefficient matrix derived from differential operators (Matsubara et al., 2020). These methods preserve known types of first integrals and suffer from temporal discretization errors. In contrast, FINDE learns any types of first integrals from data and preserves them even after temporal discretization.

The neural projection method (NPM) learns fixed holonomic constraints using the projection (and inequality constraints) (Yang et al., 2020). DGNet employed discrete gradient methods to guarantee the energy conservation in Hamiltonian systems (and the energy dissipation in friction systems) (Matsubara et al., 2020). While these methods preserve the aforementioned first integrals exactly in discrete time, their formulations are not available for other first integrals.

Several studies have proposed neural networks to learn Lyapunov functions, which are expected to be non-increasing over time, in contrast to first integrals (Manek & Kolter, 2019; Takeishi & Kawahara, 2020). If the state moves in the direction of increasing the function, it is projected onto or moved inside the contour line of the Lyapunov function. This concept is similar to that of the continuous-time version of FINDE but focuses on a single non-increasing quantity in continuous time; FINDE preserves multiple quantities in both continuous and discrete time.

3 First Integral-Preserving Neural Differential Equation

We suppose that a target system has at least K unknown functionally independent first integrals. When a neural network learns the dynamics of the target system, it is not guaranteed to learn these first integrals. We suppose that a certain neural network $\hat{f}$ for modeling the target dynamics is given, and in addition to this model $\hat{f}$, we introduce a neural network that outputs a K-dimensional vector $\boldsymbol{V}(\boldsymbol{u}) = (V_1(\boldsymbol{u})\ V_2(\boldsymbol{u})\ \dots\ V_K(\boldsymbol{u}))^\top$. Each element is expected to learn one of the first integrals as $V_k : \mathbb{R}^N \to \mathbb{R}$ for $k = 1, \dots, K$. Then, the submanifold $\mathcal{M}'$ is defined as in Eq. (1).

3.1 Continuous FINDE: Time-Derivative Projection Method

We propose a time-derivative projection method called *continuous FINDE (cFINDE)*. The cFINDE projects the time-derivative onto the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$. Roughly speaking, the cFINDE projects the dynamics on the space of the directions in which the first integrals do not change. In this way, the method can learn dynamics while preserving first integrals V, thereby finding unknown first integrals from data.

We refer to the neural network that defines the time-derivative $\hat{f} : \mathbb{R}^N \to \mathbb{R}^N$ as the base model. Applying the method of Lagrange multipliers to the projection method in Eq. (3), and taking the limit as the time-step size approaches zero, we have

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u}),\ f(\boldsymbol{u}) = \hat{f}(\boldsymbol{u}) - M(\boldsymbol{u})^\top \boldsymbol{\lambda}(\boldsymbol{u}),\ \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{V}(\boldsymbol{u}) = \boldsymbol{0}, \tag{4}$$

where $M = \frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}}$ and $\boldsymbol{\lambda} \in \mathbb{R}^N$ is the Lagrange multiplier (see Appendix B.1 for detailed derivation). We transform the second equation to obtain

$$\boldsymbol{0} = \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{V}(\boldsymbol{u}(t)) = \tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}}\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = M(\boldsymbol{u})f(\boldsymbol{u}) = M(\boldsymbol{u})(\hat{f}(\boldsymbol{u}) - M(\boldsymbol{u})^\top \boldsymbol{\lambda}(\boldsymbol{u})), \tag{5}$$

from which we obtain the Lagrange multiplier $\boldsymbol{\lambda}(\boldsymbol{u}) = (M(\boldsymbol{u})M(\boldsymbol{u})^\top)^{-1}M(\boldsymbol{u})\hat{f}(\boldsymbol{u})$. By eliminating $\boldsymbol{\lambda}(\boldsymbol{u})$, we define the cFINDE as

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u}) = (I - Y(\boldsymbol{u}))\hat{f}(\boldsymbol{u}) \text{ for } Y(\boldsymbol{u}) = M(\boldsymbol{u})^\top (M(\boldsymbol{u})M(\boldsymbol{u})^\top)^{-1} M(\boldsymbol{u}). \tag{6}$$

Theorem 1 (continuous-time first integral preservation)**.** *The cFINDE $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ preserves all first integrals V_k for $k = 1, \dots, K$ in continuous time, that is, $\frac{\mathrm{d}}{\mathrm{d}t}V_k = 0$.*

See Appendix B.1 for proof. The base model $\hat{f}$ can be a NODE, an HNN, or any other model depending on the available prior knowledge. Additionally, if a first integral is already known, it can be directly used as one of the first integrals V_k instead of being found by the neural network. Note that even though the base model $\hat{f}$ is an HNN, due to the projection, the cFINDE f is no longer a Hamiltonian system in the strict sense.

Compared to the base model $\hat{f}$, the cFINDE requires the additional computation of the neural network $\boldsymbol{V}$, several matrix multiplications, and an inverse operation. The inverse operation has a computational cost of $O(K^3)$, which is not costly if the number K of first integrals is small. Many previous models also need the inverse operation to satisfy the constraints and geometric structures, such as Lagrangian neural network (LNN) (Cranmer et al., 2020), neural symplectic form (Chen et al., 2021), and CHNN (Finzi et al., 2020b).

3.2 Discrete FINDE: Discrete-Time Derivative Projection Method

The cFINDE is still an ODE and hence needs to be solved using a numerical integrator, which causes the temporal discretization errors in the first integrals. In order to eliminate these errors, it is necessary to constrain the destination (i.e., finite difference) rather than the direction (i.e., time-derivative).

For this purpose, we propose *discrete FINDE (dFINDE)* by employing *discrete gradients* to define *discrete tangent spaces*, which are needed to constraint the state variables on the submanifold $\mathcal{M}'$.

A discrete gradient $\overline{\nabla}V$ is a discrete analogue to a gradient ∇V (Furihata & Matsuo, 2010; Gonzalez, 1996; Hong et al., 2011). Recall that a gradient ∇V of a function $V : \mathbb{R}^N \to \mathbb{R}$ can be regarded as a function $\mathbb{R}^N \to \mathbb{R}^N$ that satisfies the chain rule $\frac{\mathrm{d}}{\mathrm{d}t}V(\boldsymbol{u}) = \nabla V(\boldsymbol{u})^\top \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u}$. Analogously, a discrete gradient $\overline{\nabla}$ is defined as follows:

Definition 2 (discrete gradient). *A discrete gradient $\overline{\nabla}V$ of a function $V : \mathbb{R}^N \to \mathbb{R}$ is a function $\mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ that satisfies $V(\boldsymbol{v}) - V(\boldsymbol{u}) = \overline{\nabla}V(\boldsymbol{v}, \boldsymbol{u})^\top(\boldsymbol{v} - \boldsymbol{u})$ and $\overline{\nabla}V(\boldsymbol{u}, \boldsymbol{u}) = \nabla V(\boldsymbol{u})$.*

The first condition is a discrete analogue to the chain rule when replacing the time-derivatives $\frac{\mathrm{d}}{\mathrm{d}t}V$ and $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u}$ with finite differences $(V(\boldsymbol{v}) - V(\boldsymbol{u}))$ and $(\boldsymbol{v} - \boldsymbol{u})$, respectively, and the second condition ensures consistency with the ordinary gradient ∇V. A discrete gradient $\overline{\nabla}V$ is not uniquely determined and has been obtained manually. Recently, the automatic discrete differentiation algorithm (ADDA) has been proposed by Matsubara et al. (2020), which obtains a discrete gradient of a neural network in a manner similar to the automatic differentiation algorithm (Abadi et al., 2016; Paszke et al., 2017). The discrete gradient is defined in discrete time; hence, the prediction using the discrete gradient is free from temporal discretization errors. See Appendix B.2 and the references Furihata & Matsuo (2010); Matsubara et al. (2020) for more details.

Following Christiansen et al. (2011); Dahlby et al. (2011), we introduce a discrete analogue to the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$ called the discrete tangent space $\mathcal{T}_{(\boldsymbol{v},\boldsymbol{u})}\mathcal{M}'$. In particular, for a pair of points $(\boldsymbol{v}, \boldsymbol{u}) \in \mathcal{M}'$, the discrete tangent space is defined as

$$\mathcal{T}_{(\boldsymbol{v},\boldsymbol{u})}\mathcal{M}' = \{\boldsymbol{w} \in \mathbb{R}^N : \overline{\nabla}V_k(\boldsymbol{v}, \boldsymbol{u})^\top \boldsymbol{w} = 0 \text{ for } k = 1, \dots, K\}. \tag{7}$$

If the finite difference $(\boldsymbol{u}^{s+1} - \boldsymbol{u}^s)$ between the predicted and current states is on the discrete tangent space $\mathcal{T}_{(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)}\mathcal{M}'$, the first integrals V_k are preserved because $V_k(\boldsymbol{u}^{s+1}) - V_k(\boldsymbol{u}^s) = \overline{\nabla}V_k(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)^\top(\boldsymbol{u}^{s+1} - \boldsymbol{u}^s) = 0$. Note that similar concepts defined in different ways are also referred to as discrete tangent spaces (Cuell & Patrick, 2009; Dehmamy et al., 2021).

We suppose that a neural network (e.g., NODE) $\hat{f}$ defines an ODE and a numerical integrator predicts the next state $\tilde{\boldsymbol{u}}^{s+1}$ from a given state $\boldsymbol{u}^s$. We call this process a discrete-time base model $\hat{\psi}$, which satisfies $\frac{\tilde{\boldsymbol{u}}^{s+1} - \boldsymbol{u}^s}{\Delta t^s} = \hat{\psi}(\boldsymbol{u}^s; \Delta t^s)$. Subsequently, we consider the model

$$\begin{aligned} &\tfrac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s; \Delta t^s), \\ &\psi(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s; \Delta t^s) = \hat{\psi}(\boldsymbol{u}^s; \Delta t^s) - \overline{M}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)^\top \boldsymbol{\lambda}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s),\ \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) = \boldsymbol{0}, \end{aligned} \tag{8}$$

where $\overline{M}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s) = (\overline{\nabla}V_1(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)\ \dots\ \overline{\nabla}V_K(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s))^\top$. As shown in Appendix B.1, this formulation is also derived from the projection method in Eq. (3). Using the chain rule of the discrete gradient,

$$\boldsymbol{0} = \tfrac{\boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s)}{\Delta t^s} = \overline{M}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)\tfrac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^s}{\Delta t^s} = \overline{M}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)\psi(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s; \Delta t^s), \tag{9}$$

Substituting this into Eq. (8) and eliminating the Lagrange multiplier $\boldsymbol{\lambda}$, we define the dFINDE as

$$\tfrac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s; \Delta t^s) = (I - \overline{Y}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s))\hat{\psi}(\boldsymbol{u}^s; \Delta t^s) \text{ for } \overline{Y} = \overline{M}^\top(\overline{M}\,\overline{M}^\top)^{-1}\overline{M}, \tag{10}$$

where we have abbreviated $\overline{M}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)$ and $\overline{Y}(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s)$ to $\overline{M}$ and $\overline{Y}$, respectively.

Theorem 2 (discrete-time first integral preservation). *The dFINDE $\frac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s; \Delta t^s)$ preserves all first integrals V_k for $k = 1, \dots, K$ in discrete time, that is, $V_k(\boldsymbol{u}^{s+1}) - V_k(\boldsymbol{u}^s) = 0$.*

See Appendix B.1 for proof. Intuitively, dFINDE projects the finite difference (discrete-time derivative) $\hat{\psi}$ onto the discrete tangent space $\mathcal{T}_{(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)}\mathcal{M}'$ after the numerical integration for each step, whereas cFINDE projects the time-derivative $\hat{f}$ onto the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$ at every substep inside a numerical integrator. In the discrete-time base model $\hat{\psi}$, the ODE $\hat{f}$ can be defined by any model, such as NODE or HNN, and the numerical integrator can be implemented by any method, such as the Runge–Kutta method or the leapfrog integrator. The projection method in Eq. (3), the method in Eq. (8), and the dFINDE in Eq (10) are implicit methods and hence relatively computationally expensive. However, only the dFINDE can be trained non-iteratively by standard backpropagation algorithms. As explained in Appendix B.3, this is because the next state $\boldsymbol{u}^{s+1}$ is given during training and the ADDA can explicitly obtain the discrete gradient and its computational graph.

Table 2: Datasets, Dynamics, and First Integrals.

			First Integrals			
Dataset	**Dynamics (Structure)**	N	**Energy**	**Momentum**	**Mass**	**Constraint**
Two-body problem	Canonical Hamiltonian	8	✓	✓		
Discretized KdV equation	Non-canonical Hamiltonian	50	✓		✓	
Double pendulum	Poisson	8	✓			✓
FitzHugh–Nagumo model	Dirac	4				✓

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Target Systems We evaluated FINDE and base models using datasets associated with first integrals; these are summarized in Table 2. A gravitational two-body problem (2-body) on a 2-dimensional configuration space is a typical Hamiltonian system in the canonical form. In addition to the total energy, the system has first integrals related to symmetries in space, namely, the linear and angular momenta. The Korteweg–De Vries (KdV) equation is a PDE model of shallow water waves. This equation is a Hamiltonian system in a non-canonical form and has the Hamiltonian, total mass, and many other quantities as first integrals. We discretized the KdV equation in space, obtaining a fifty-dimensional state $\boldsymbol{u}$. A double pendulum (2-pend) is a Hamiltonian system in polar coordinates. However, we transformed it to Cartesian coordinates; hence, it became a Poisson system. The lengths of the two rods work as holonomic constraints and lead to four first integrals in addition to the Hamiltonian. The FitzHugh–Nagumo model is a biological neuron model as an electric circuit, which exhibits a rapid and transient change of voltage called a spike. As an electric circuit, the currents through and voltages applied to the inductor and capacitor can be regarded as system states, which are constrained by the circuit topology and Kirchhoff's current and voltage laws. Then, this system has a state of four elements and two first integrals. Because the resistor dissipates the energy, the system is not a Poisson system, but a Dirac structure can be found (van der Schaft & Jeltsema, 2014). We generated a time-series set of each dataset with different initial conditions (hence, different values of first integrals). See Appendix C for more details.

Implementation We implemented the proposed FINDE and evaluated it under the following settings. We implemented all codes by modifying the officially released codes of HNN (Greydanus et al., 2019) [1] and DGNet (Matsubara et al., 2020)[2]. We used Python v. 3.8.12 with packages scipy v. 1.7.3, pytorch v. 1.10.2, torchdiffeq v. 0.1.1, functorch v. 1.10 preview, and gplearn v. 0.4.2. We used the Dormand–Prince method (dopri5) (Dormand & Prince, 1986) as the numerical integrator, except in Section 4.2. All experiments were performed on a single NVIDIA A100.

Following HNN (Greydanus et al., 2019) and DGNet (Matsubara et al., 2020), we used fully-connected neural networks with two hidden layers. The input was the state $\boldsymbol{u}$, and the output represented the first integrals $\boldsymbol{V}$ for FINDE, time-derivative $\hat{f}$ for NODE, or the Hamiltonian H for HNN. Each hidden layer had 200 units and preceded a hyperbolic tangent activation function. Each weight matrix was initialized as an orthogonal matrix. For the KdV dataset, we used a 1-dimensional CNN, wherein the kernel size of each layer was 3. The double pendulum is a second–order system, implying that the time-derivative $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q}$ of the position $\boldsymbol{q}$ is known to be the velocity $\boldsymbol{v}$. Hence, we treated only the acceleration $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}$ as the output to learn in the 2-pend dataset. This assumption slightly improved the absolute performances but did not change the relative trends.

As the loss function for the cFINDE, we used the mean squared error (MSE) between the ground truth future state $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ and the future state $\boldsymbol{u}_{\mathrm{pred.}}^{s+1}$ predicted from the current step $\boldsymbol{u}_{\mathrm{GT}}^{s}$ normalized by the time-step size Δt^s; we named this the *1-step error*. For the dFINDE, we used the MSE between the left- and right-hand sides of Eq. (10) because the ground truth states $\boldsymbol{u}_{\mathrm{GT}}^{s}$ and $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ are available during the training phase. The base model and FINDE were jointly trained using the Adam optimizer (Kingma & Ba, 2015) with the parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ and a batch size of 200.

[1]`https://github.com/greydanus/hamiltonian-nn`

[2]`https://github.com/tksmatsubara/discrete-autograd`

The learning rate was initialized to 10^{-3} and decayed to zero with cosine annealing (Loshchilov & Hutter, 2017). See Appendix B.3 and the enclosed source code for details about implementations.

Evaluation Metric We used the 1-step error as an evaluation metric, which is identical to the loss function for the cFINDE, and displayed it in the scale $\times 10^{-9}$. The lower this indicator, the better, as indicated by $\downarrow$. The MSEs of the state or system energy over a long period are misleading indicators, as suggested in prior studies (Botev et al., 2021; Jin et al., 2020b; Vlachas et al., 2020). For example, a periodic orbit that is correctly learned except for a slight difference in angular velocity would have the same MSE as an orbit that never moves from its initial position. Instead, we used the valid prediction time (*VPT*) (Botev et al., 2021; Jin et al., 2020b; Vlachas et al., 2020). VPT denotes the time point s divided by the length S of the time-series at which the MSE of the predicted state $\boldsymbol{u}^s_{\text{pred.}}$ first exceeds a given threshold θ in an initial value problem, that is,

$$VPT(\boldsymbol{u}_{\text{pred.}}; \boldsymbol{u}_{\text{GT}}) = \tfrac{1}{S}\max\{s_f | \text{MSE}(\boldsymbol{u}^s_{\text{pred.}}, \boldsymbol{u}^s_{\text{GT}}) < \theta \text{ for all } s \le s_f,\ 0 \le s_f \le S\}. \quad (11)$$

The higher this indicator, the better, as indicated by $\uparrow$. To obtain VPTs, we normalized each element of state to have the zero mean and unit variance in the training data and set θ to 0.01. For systems with "spiking" behaviors, a small error in phase may be regarded as a significant error in the state; for the FitzHugh–Nagumo model, we obtained the VPTs by allowing for a delay and advance of up to 5 steps.

4.2 Demonstration of First Integral Preservation

Before learning first integrals from data, we demonstrate that dFINDE can preserve first integrals without temporal discretization errors. We used a mass-spring system, which had the state $\boldsymbol{u} = (q\ v)^\top$, dynamics $\frac{\mathrm{d}}{\mathrm{d}t}q = v$ and $\frac{\mathrm{d}}{\mathrm{d}t}v = -q$, and system energy $E(q, v) = \frac{1}{2}(q^2 + v^2)$. Using an initial value of $(1.0\ \ 0.0)^\top$ and a time-step size of $\Delta t = 0.2$, we solved the initial value problem of the true ODE using the leapfrog integrator with or without FINDE, with the true system energy E as the first integral V. Notably, no neural networks nor training were involved.

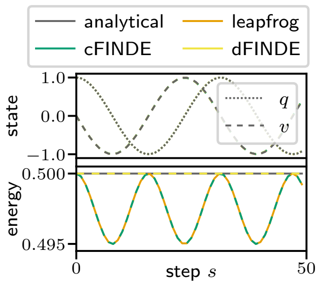

Figure 1: Integration of a known mass-spring system by the leapfrog integrator. (top) States predicted by comparison methods. (bottom) Energy calculated from the states predicted.

Figure 1 shows the results, along with the analytical solution. The states predicted by comparison methods overlap and are apparently identical. However, the energy obtained by the leapfrog integrator fluctuates and the same is true for cFINDE. This is because the leapfrog integrator and cFINDE suffer from temporal discretization errors in first integrals. In contrast, dFINDE preserves the energy accurately, the same as the analytical solution. This is because dFINDE projects the state $(q\ v)^\top$ onto the discrete tangent space $\mathcal{T}_{(\boldsymbol{v},\boldsymbol{u})}\mathcal{M}'$ at every step. Although a smaller time-step size reduces temporal discretization errors, this result demonstrates the advantage of dFINDE. See Appendix D.1 for the case with the Dormand-Prince integrator.

4.3 Finding Non-Hamiltonian First Integrals of Hamiltonian Systems

We evaluated cFINDE and dFINDE on learning from the 2-body dataset. We used HNN as the base model $\hat{f}$. We found that cFINDE and dFINDE obtained better performances if it did not treat the Hamiltonian H of the HNN as one of the first integrals V_k. The medians and standard deviations of five trials are summarized in the leftmost column of Table 3. The cFINDE achieved better VPTs than the original HNN with $K = 1$ to 2, and its performance was suddenly degraded with $K = 3$. The dFINDE showed a similar trend with slightly better performances; there is a trade-off between performance and computational cost. The HNN with either cFINDE or dFINDE found two first integrals in addition to the Hamiltonian H of the HNN. Even though a two-body problem is a Hamiltonian system that an HNN can learn, the prior knowledge that there exist first integrals other than the Hamiltonian H can be a clue that enables better learning. Despite their better long-term prediction performance, the HNN with either cFINDE or dFINDE yielded 1-step errors worse than the HNN, indicating that the 1-step error is misleading as an evaluation criterion.

These example results are depicted in Fig. 2. In the absence of FINDE, the mass positions (x_1, y_1) and (x_2, y_2) became inaccurate in a short time and the center-of-gravity position $(x_c, y_c) =$

Table 3: Results of cFINDE and dFINDE.

		2-body + HNN		KdV		2-pend		FitzHugh–Nagumo	
Model	K	**1-step↓**	**VPT↑**	**1-step↓**	**VPT↑**	**1-step↓**	**VPT↑**	**1-step↓**	**VPT↑**
base model	–	5.17 ±0.57	0.362 ±0.026	5.59 ±0.30	0.339 ±0.038	0.82 ±0.02	0.110 ±0.035	73.66 ±12.59	0.236 ±0.053
+ cFINDE	1	7.10 ±1.25	$\underline{0.374}$ ±0.036	6.24 ±0.44	$\underline{0.371}$ ±0.088	$\underline{0.75}$ ±0.04	$\underline{0.156}$ ±0.042	$\underline{54.18}$ ±8.12	0.127 ±0.148
	2	7.78 ±1.39	$\underline{\mathbf{0.450}}$ ±0.052	$\underline{\mathbf{2.59}}$ ±0.11	$\underline{0.608}$ ±0.085	$\underline{0.73}$ ±0.05	$\underline{0.198}$ ±0.088	$\underline{\mathbf{37.03}}$ ±3.81	$\underline{\mathbf{0.437}}$ ±0.084
	3	$>10^3$	0.147 ±0.146*	$\underline{3.19}$ ±0.37	$\underline{\mathbf{0.730}}$ ±0.091	$\underline{\mathbf{0.69}}$ ±0.03	$\underline{0.411}$ ±0.093	$>10^6$	0.007 ±0.007*
	4	$>10^3$	0.101 ±0.005	$\underline{3.65}$ ±0.30	$\underline{0.641}$ ±0.071	$\underline{0.77}$ ±0.07	$\underline{0.395}$ ±0.083	—	
	5	$>10^3$	0.080 ±0.014	$\underline{4.68}$ ±0.43	$\underline{0.601}$ ±0.069	$\underline{0.80}$ ±0.07	$\underline{\mathbf{0.585}}$ ±0.097	—	
	6	$>10^3$	0.070 ±0.019	7.79 ±0.51	$\underline{0.425}$ ±0.067	12.53 ±0.00	0.005 ±0.000*	—	
+ dFINDE	1	7.01 ±1.06	$\underline{0.379}$ ±0.040	11.61 ±6.60	0.288 ±0.083	$\underline{0.75}$ ±0.10	$\underline{0.152}$ ±0.017	$\underline{47.07}$ ±8.03	0.117 ±0.122
	2	7.03 ±1.00	$\underline{\mathbf{0.475}}$ ±0.022	$\underline{\mathbf{2.70}}$ ±0.26	$\underline{0.598}$ ±0.059	$\underline{0.74}$ ±0.05	$\underline{0.271}$ ±0.111	$\underline{\mathbf{33.24}}$ ±3.40	$\underline{\mathbf{0.455}}$ ±0.032
	3	54.78 ±36.39	0.309 ±0.024	$\underline{3.78}$ ±0.27	$\underline{0.636}$ ±0.024	$\underline{\mathbf{0.69}}$ ±0.05	$\underline{0.447}$ ±0.081	319.70 ±91.11	0.049 ±0.007
	4	$>10^3$	0.102 ±0.015	$\underline{3.48}$ ±0.32	$\underline{\mathbf{0.780}}$ ±0.059	$\underline{0.71}$ ±0.03	$\underline{0.454}$ ±0.060	—	
	5	$>10^3$	0.086 ±0.011*	$\underline{5.26}$ ±0.15	$\underline{0.718}$ ±0.038	0.86 ±0.09	$\underline{\mathbf{0.591}}$ ±0.087	—	
	6	$>10^3$	0.059 ±0.017	9.60 ±3.61	$\underline{0.573}$ ±0.121	58.88 ±22.98	0.037 ±0.039	—	

Notes: Standard deviation follows the $\pm$ symbol; underlined results are better than those of the base models; bold font indicates best results; * denotes trials that failed in training because of underflow of time-step size.

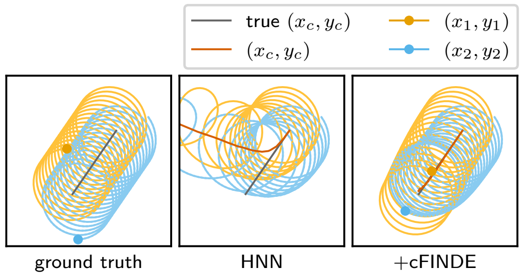

Figure 2: Example results of 2-body dataset. (left) Ground truth. (middle) HNN. (right) HNN with cFINDE.

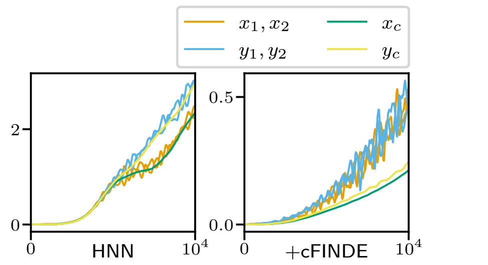

Figure 3: Mean absolute errors for results of 2-body dataset. (left) HNN. (right) HNN with cFINDE.

$(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ deviated rapidly. The HNN with cFINDE accurately predicted the state for a longer period. Even after errors in the mass positions became non-negligible, errors in the center-of-gravity position were still small. Figure 3 shows the absolute errors averaged over all trials, which demonstrate how the trend changes with cFINDE. In both the x- and y-directions, the HNN without FINDE produced errors in the center-of-gravity position x_c (or y_c), and those in the mass positions x_1, x_2 (or y_1, y_2) at a similar level. In contrast, with the cFINDE, errors in the center-of-gravity position were much smaller than those in the mass positions, implying that errors in one mass position canceled out errors in the other. We performed a symbolic regression of first integrals $\boldsymbol{V}$ found by the neural network. For $K = 2$, the found first integrals $\boldsymbol{V}$ were identical to the linear momenta in the x- and y-directions up to affine transformation in most cases. See Appendix D.2 for detailed results. Therefore, we conclude that FINDE not only had better prediction accuracy but also found and preserved linear momenta (which are related to symmetries in space) more accurately despite not having prior knowledge about symmetries.

4.4 Finding First Integrals of Unknown Systems

It is often unclear whether a target system is a Hamiltonian system or not, but one can expect that it has several first integrals. We evaluated cFINDE and dFINDE using NODE as the base model and display the results in Table 3.

For the KdV dataset, the NODE with either cFINDE or dFINDE obtained improved VPTs for a wide range of K. Figure 4 shows an example result. The prediction states were apparently similar. In the absence of FINDE, the NODE increased all of its errors in proportion to time. With cFINDE, the error in total mass increased at the point where the two solitons collided, but then returned to the original level. Although the calculation is slightly inaccurate, the cFINDE learned to preserve the total mass. The error in energy continued to increase for $K = 2$, but remained within a small range for $K = 3$. These results suggest that the first or second quantity learned by the cFINDE was total

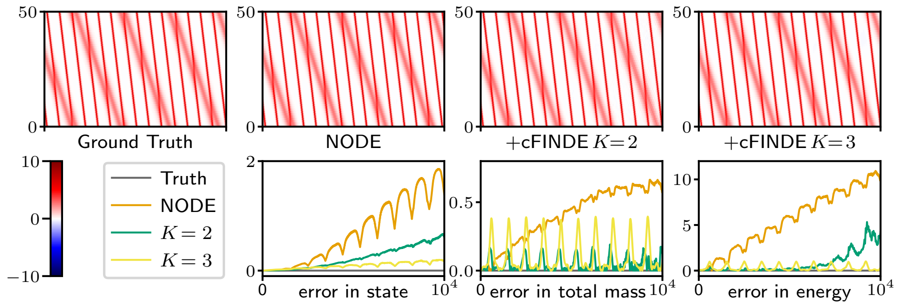

Figure 4: Example results of KdV dataset. (top) Predicted states. Red belts denote moving solitons. (bottom) Mean absolute errors in states $\boldsymbol{u}$, total mass $\sum_{k=1}^{N} u_k$, and energy, from left to right.

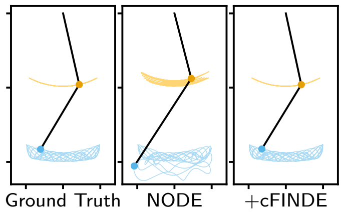

Figure 5: Example results of 2-pend dataset for 2,000 steps. (left) Ground truth. (middle) NODE. (right) NODE with cFINDE for $K = 5$.

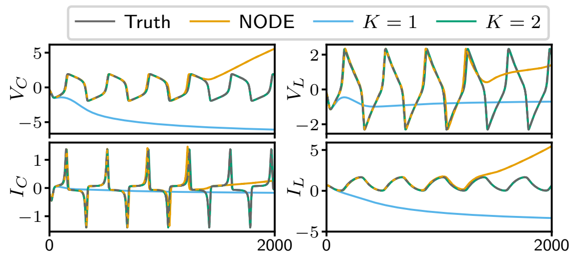

Figure 6: Example results of FitzHugh–Nagumo dataset. Each panel shows one of four states.

mass, the third quantity was system energy, and the remaining quantity may correspond to one of the many first integrals of the KdV equation.

For the 2-pend dataset, the NODE with either cFINDE or dFINDE obtained improved VPTs with $K = 1$ to 5. In addition to the system energy, the double pendulum has two holonomic constraints on the position, which lead to two additional constraints involving the velocity (see Appendix C for details). Thus, it is reasonable that the NODE with either cFINDE or dFINDE obtained the best VPT for $K = 5$ first integrals and completely failed for $K > 5$ first integrals. As exemplified in Fig. 5, the NODE without FINDE did not preserve the lengths of rods, making the states deviate gradually. See Appendix D.3 for the case when actual constraints are known. For the FitzHugh–Nagumo dataset, the NODE with either cFINDE or dFINDE obtained improved VPTs for $K = 2$. As exemplified in Fig. 6, the ground truth state converged to a periodic orbit, and only the NODE with cFINDE for $K = 2$ reproduced similar dynamics. Without FINDE, the state did not remain in a limited region. For $K = 1$, the state converged to a wrong equilibrium; the sole quantity V_1 may have attempted and failed to learn both first integrals. We conclude that both cFINDE and dFINDE found all first integrals of the 2-pend and FitzHugh–Nagumo datasets; $K = 5$ and $K = 2$, respectively.

5 Conclusion

This study proposed *first integral-preserving neural differential equation* (FINDE), which can find and preserve any type of first integrals from data in a unified manner. FINDE projects the time evolution onto the submanifold defined using the (discrete) gradients of first integrals represented by a neural network. We experimentally demonstrated that FINDE found and preserved first integrals that come from the energy and mass conservation laws, symmetries in space, and constraints, thereby predicting the dynamics for far longer. FINDE is available even for an energy-dissipating system. When FINDE obtains the best prediction accuracy with $K = K'$, it suggests that the target system has at least K' first integrals. Hence, FINDE has the potential to make scientific discoveries by revealing geometric structures of dynamical systems. See Appendix D.4 for more discussions on K.

The numerical error tolerance 10^{-9} was negligible compared to the 1-step errors (which were 10^{-5} to 10^{-4} in absolute error). However, the dFINDE tended to obtain much better VPTs than the cFINDE. This result suggests that a method leading to smaller numerical errors produces a model with smaller modeling errors, as observed in previous works (Chen et al., 2020; Matsubara et al., 2020). These results may form a new frontier for integrating numerical and modeling errors.

Reproducibility Statement

See Section 4.1 for experimental settings. More detailed descriptions can be found in Appendix B.3 for training procedure and Appendix C for datasets. The authors have enclosed the source code for generating the datasets and running the experiments as supplementary material.

Acknowledgement

This study was partially supported by JST CREST (JPMJCR1914), JST PRESTO (JPMJPR21C7), and JSPS KAKENHI (19K20344, 20K11693).

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

Ferran Alet, Dylan Doblar, Allan Zhou, Joshua Tenenbaum, Kenji Kawaguchi, and Chelsea Finn. Noether Networks: Meta-Learning Useful Conserved Quantities. *Advances in Neural Information Processing Systems (NeurIPS)*, (NeurIPS):1–20, 2021.

David G. T. Barrett and Benoit Dherin. Implicit Gradient Regularization. In *International Conference on Learning Representations (ICLR)*, 2021.

Aleksandar Botev, Andrew Jaegle, Peter Wirnsberger, Daniel Hennes, and Irina Higgins. Which priors matter? Benchmarking models for learning latent dynamics. In *Advances in Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

Yuan Cao, Zhiying Fang, Yue Wu, Ding Xuan Zhou, and Quanquan Gu. Towards Understanding the Spectral Bias of Deep Learning. *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2205–2211, 2021.

Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren. Learning Hamiltonians of Constrained Mechanical Systems. *arXiv*, pp. 1–18, 2022.

S. Chen, S. A. Billings, and P. M. Grant. Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud, Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1–19, 2018.

Yuhan Chen, Takashi Matsubara, and Takaharu Yaguchi. Neural Symplectic Form : Learning Hamiltonian Equations on General Coordinate Systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on Learning Representations (ICLR)*, pp. 1–23, 2020.

Snorre H. Christiansen, Hans Z. Munthe-Kaas, and Brynjulf Owren. Topics in structure-preserving discretization. *Acta Numerica*, 20:1–119, 2011.

D S Clouse, C L Giles, B G Horne, and G W Cottrell. Time-delay neural networks: Representation and induction of finite-state machines. *IEEE Transactions on Neural Networks*, 8(5):1065–70, January 1997.

Kevin L. Course, Trefor W. Evans, and Prasanth B. Nair. Weak form generalized Hamiltonian learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, number NeurIPS, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian Neural Networks. In *ICLR Deep Differential Equations Workshop*, pp. 1–9, March 2020.

Charles Cuell and George W. Patrick. Geometric discrete analogues of tangent bundles and constrained Lagrangian systems. *Journal of Geometry and Physics*, 59(7):976–997, 2009.

Morten Dahlby, Brynjulf Owren, and Takaharu Yaguchi. Preserving multiple first integrals by discrete gradients. *Journal of Physics A: Mathematical and Theoretical*, 44(30), 2011.

Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic Symmetry Discovery with Lie Algebra Convolutional Network. In *Advances in Neural Information Processing Systems (NeurIPS)*, number 2018, pp. 1–30, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*, pp. 1–15, October 2018.

J. R. Dormand and P. J. Prince. A reconsideration of some embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 15(2):203–211, 1986.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data. In *International Conference on Machine Learning (ICML)*, pp. 3146–3157, 2020a.

Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. Simplifying Hamiltonian and Lagrangian Neural Networks via Explicit Constraints. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020b.

Marc Finzi, Max Welling, and Andrew Gordon Wilson. A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups. In *International Conference on Machine Learning (ICML)*, 2021.

K. Fukunaga and D.R. Olsen. An Algorithm for Finding Intrinsic Dimensionality of Data. *IEEE Transactions on Computers*, C-20(2):176–183, February 1971.

Daisuke Furihata. A stable and conservative finite difference scheme for the Cahn-Hilliard equation. *Numerische Mathematik*, 87(4):675–699, February 2001.

Daisuke Furihata and Takayasu Matsuo. *Discrete Variational Derivative Method: A Structure-Preserving Numerical Method for Partial Differential Equations*. Chapman and Hall/CRC, December 2010.

C. W. Gear. Maintaining Solution Invariants in the Numerical Solution of ODE s. *SIAM Journal on Scientific and Statistical Computing*, 7(3):734–743, 1986.

O. Gonzalez. Time integration and discrete Hamiltonian systems. *Journal of Nonlinear Science*, 6 (5):449–467, September 1996.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1–16, 2019.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31. Springer-Verlag, Berlin/Heidelberg, 2006.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, December 2016.

Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations (ICLR)*, 2020.

Jialin Hong, Shuxing Zhai, and Jingjing Zhang. Discrete Gradient Approach to Stochastic Differential Equations with a Conserved Quantity. *SIAM Journal on Numerical Analysis*, 49(5): 2017–2038, January 2011.

Eugene M. Izhikevich and Richard FitzHugh. FitzHugh-Nagumo model. http://scholarpedia.org/article/FitzHugh-Nagumo_model, 2006.

Pengzhan Jin, Zhen Zhang, Ioannis G. Kevrekidis, and George Em Karniadakis. Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks. pp. 1–12, 2020a.

Pengzhan Jin, Aiqing Zhu, George Em Karniadakis, and Yifa Tang. Symplectic networks: Intrinsic structure-preserving networks for identifying Hamiltonian systems. *Neural Networks*, 132:166–179, 2020b.

Muhammad Firmansyah Kasim and Yi Heng Lim. Constants of motion network, August 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, pp. 1–15, December 2015.

Kookjin Lee and Kevin Carlberg. Deep Conservation: A latent-dynamics model for exact satisfaction of physical conservation laws. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Asriel U. Levin and Kumpati S. Narendra. Recursive identification using feedforward neural networks. *International Journal of Control*, 61(3):533–547, 1995.

Ziming Liu and Max Tegmark. Machine Learning Conservation Laws from Trajectories. *Physical Review Letters*, 126(18):180604, May 2021.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-Net: Learning PDEs from Data. In *International Conference on Machine Learning (ICML)*, pp. 3208–3216. PMLR, July 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, pp. 1–16, 2017.

Gaurav Manek and J. Zico Kolter. Learning Stable Deep Dynamics Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1–9, 2019.

Takashi Matsubara, Ai Ishikawa, and Takaharu Yaguchi. Deep Energy-Based Modeling of Discrete-Time Physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Robert M. Miura, Clifford S. Gardner, and Martin D. Kruskal. Korteweg-de Vries equation and generalizations. II. Existence of conservation laws and constants of motion. *Journal of Mathematical Physics*, 9(8):1204–1209, 1968.

Kumpati S. Narendra and Kannan Parthasarathy. Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

Oliver Nelles. *Nonlinear System Identification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. Automatic differentiation in PyTorch. In *Autodiff Workshop on Advances in Neural Information Processing Systems*, pp. 1–4, 2017.

Florian A. Potra and Jeng Yen. Implicit numerical integration for euler-lagrange equations via tangent space parametrization. *Mechanics of Structures and Machines*, 19(1):77–98, 1991.

Lionel Raff, Ranga Komanduri, Martin Hagan, and Satish Bukkapatnam. *Neural Networks in Chemical Reaction Dynamics*. 2012.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Stephan Rasp, Peter D. Dueben, Sebastian Scher, Jonathan A. Weyn, Soukayna Mouatadid, and Nils Thuerey. WeatherBench: A Benchmark Data Set for Data-Driven Weather Forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11), 2020.

Akiyoshi Sannai, Masaaki Imaizumi, and Makoto Kawano. Improved Generalization Bounds of Group Invariant / Equivariant Deep Networks via Quotient Feature Spaces. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, October 2021.

Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of GANs for semantic face editing. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 9240–9249, 2020.

J. Sjöberg, H. Hjalmarsson, and L. Ljung. Neural Networks in System Identification. *IFAC Proceedings Volumes*, 27(8):359–382, 1994.

Yifan Sun, Linan Zhang, and Hayden Schaeffer. NeuPDE: Neural Network Based Ordinary and Partial Differential Equations for Modeling Time-Dependent Data. In *Mathematical and Scientific Machine Learning Conference*, pp. 352–372. PMLR, August 2020.

Naoya Takeishi and Yoshinobu Kawahara. Learning dynamics models with stable invariant sets. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Takeshi Teshima, Koichi Tojo, Masahiro Ikeda, Isao Ishikawa, and Kenta Oono. Universal Approximation Property of Neural Ordinary Differential Equations. In *NeurIPS Workshop on Differential Geometry Meets Deep Learning (DiffGeo4DL)*, 2020.

Ricardo M. Trigo and Jean P. Palutikof. Simulation of daily temperatures for climate change scenarios over Portugal: A neural network model approach. *Climate Research*, 13(1):45–59, 1999.

Arjan van der Schaft and Dimitri Jeltsema. Port-Hamiltonian Systems Theory: An Introductory Overview. *Foundations and Trends® in Systems and Control*, 1(2):173–378, 2014.

P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and Reservoir Computing in Recurrent Neural Networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.

Yi Jen Wang and Chin Teng Lin. Runge-Kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2):294–307, 1998.

Shuqi Yang, Xingzhe He, and Bo Zhu. Learning Physical Constraints with Neural Projections. *Advances in Neural Information Processing Systems*, pp. 1–15, 2020.

Ge Zhong and Jerrold E Marsden. Lie-Poisson Hamilton-Jacobi Theory and Lie-Poisson Integrators. *Physics Letters A*, 133(3):3–8, November 1988.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *International Conference on Learning Representations (ICLR)*, pp. 1–17, 2020a.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning. *arXiv*, pp. 1–6, 2020b.

A HAMILTONIAN SYSTEM, ITS GENERALIZATION, AND FIRST INTEGRALS

Preliminary In this section, we briefly introduce potential target systems and related works. Methods proposed by related works use specific prior knowledge about target systems, such as constraints. In contrast, our proposed FINDE assumes a situation where neural networks learn systems with unknown properties. See, for example, Hairer et al. (2006); van der Schaft & Jeltsema (2014) for more details about geometric mechanics.

On an N-dimensional manifold $\mathcal{M}$, an ODE is defined using a vector field $f : \mathcal{M} \to \mathcal{T}_{\boldsymbol{u}}\mathcal{M}$, which maps a point $\boldsymbol{u}$ on the manifold $\mathcal{M}$ to a tangent vector $f(\boldsymbol{u})$ on the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}$. The NODE defines an ODE in this way (Chen et al., 2018). Given a scalar-valued function $H : \mathcal{M} \to \mathbb{R}$ on the manifold $\mathcal{M}$, its differential $\mathrm{d}H : \mathcal{M} \to \mathcal{T}^*_{\boldsymbol{u}}\mathcal{M}$ is a cotangent vector field (a.k.a. a differential 1-form), which maps a point $\boldsymbol{u}$ on the manifold $\mathcal{M}$ to a cotangent vector $\mathrm{d}H(\boldsymbol{u})$ on the cotangent space $\mathcal{T}^*_{\boldsymbol{u}}\mathcal{M}$.

Hamiltoanian System A Hamiltonian system is defined using a non-degenerate closed differential 2-form ω called symplectic form, which is a skew-symmetric bilinear map $\omega_{\boldsymbol{u}} : \mathcal{T}_{\boldsymbol{u}}\mathcal{M} \times \mathcal{T}_{\boldsymbol{u}}\mathcal{M} \to \mathbb{R}$ at point $\boldsymbol{u}$. A symplectic form assigned to a manifold is called the symplectic structure. The coordinate-free form of Hamilton's equation is $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = X_H(\boldsymbol{u})$, $\omega_{\boldsymbol{u}}(X_H(\boldsymbol{u}), \boldsymbol{w}) = \langle \mathrm{d}H(\boldsymbol{u}), \boldsymbol{w}\rangle$ for any $\boldsymbol{w} \in \mathcal{T}_{\boldsymbol{u}}\mathcal{M}$, where X_H is the Hamiltonian vector field. The symplectic form ω gives rise to a bundle map $\omega^\flat_{\boldsymbol{u}} : \mathcal{T}_{\boldsymbol{u}}\mathcal{M} \to \mathcal{T}^*_{\boldsymbol{u}}\mathcal{M}$, with which Hamilton's equation is rewritten as $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = X_H(\boldsymbol{u}) = (\omega^\flat_{\boldsymbol{u}})^{-1}(\mathrm{d}H(\boldsymbol{u}))$. The right-hand side is locally equivalent to the product of a coefficient matrix S and the gradient ∇H of the Hamiltonian H. Then, Hamilton's equation is obtained as $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = S\nabla H(\boldsymbol{u})$. Hamiltonian systems are often expressed in the canonical form, in other words, they are defined on Darboux coordinates, on which the state $\boldsymbol{u}$ is the paired generalized position $\boldsymbol{q}$ and generalized momentum $\boldsymbol{p}$. The corresponding coefficient matrix is $S = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$ for $2n = N$ and the n-dimensional identity matrix I_n. The HNN was developed to model Hamiltonian systems in the canonical forms (Greydanus et al., 2019).

An Euler–Lagrange equation with a hyperregular Lagrangian and a Lotka–Volterra equation are also Hamiltonian systems; however, their coordinate systems are not Darboux coordinates. A neural symplectic form (NSF) handles this class of equations (Chen et al., 2021). The KdV equation is also a Hamiltonian system not on Darboux coordinates. For Hamiltonian PDE systems, HNN++ was proposed (Matsubara et al., 2020). According to Darboux's theorem, any Hamiltonian system on an even–dimensional manifold can be transformed into the canonical form.

Noether's theorem states that a continuous symmetry of a system leads to a conservation law. A Hamiltonian system is symmetric (invariant) to translation in time and conserves the Hamiltonian H. A two-body problem is symmetric to translation and rotation in space and conserves linear and angular momenta. These quantities are first integrals. LieConv and EMLP-HNN had such symmetries implemented in their architectures (Finzi et al., 2020a; 2021). A pendulum is not symmetric to translation and rotation in space and does not conserve linear and angular momenta, but does exchange them with the base to which it is fixed.

Poisson System A Poisson system is named after a Poisson bracket $\{\cdot,\cdot\}$, but it is convenient to refer to it as a degenerate Hamiltonian system. A Poisson bracket is defined using a Poisson 2-vector B, which is a skew-symmetric bilinear map $B_{\boldsymbol{u}} : \mathcal{T}^*_{\boldsymbol{u}}\mathcal{M} \times \mathcal{T}^*_{\boldsymbol{u}}\mathcal{M} \to \mathbb{R}$ at point $\boldsymbol{u}$. The Poisson 2-vector B gives rise to a bundle map $B^\sharp_{\boldsymbol{u}} : \mathcal{T}^*_{\boldsymbol{u}}\mathcal{M} \to \mathcal{T}_{\boldsymbol{u}}\mathcal{M}$ and defines Hamilton's equation as $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = B^\sharp(\mathrm{d}H(\boldsymbol{u}))$. The Darboux–Lie theorem states that any Poisson system can be transformed into the canonical form $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = S\nabla H(\boldsymbol{u})$ by using a matrix $S = \begin{pmatrix} 0 & I_k & 0 \\ -I_k & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ for $2k < N$. The last $N - 2k$ elements remain unchanged and correspond to the first integrals. In this sense, a Poisson system is a degenerate Hamiltonian system. A Poisson 2-vector assigned to a manifold is called a Poisson structure. Several models of the dynamics of disease spreading and chemical reactions are Poisson systems, and total population and molecular mass are typical first integrals.

A Poisson neural network (PNN) learns to transform a given Poisson system into a canonical form (Jin et al., 2020a).

Constrained Hamiltonian System A constraint $C(\boldsymbol{q}) = 0$ on the position $\boldsymbol{q}$ is called a holonomic constraint. Holonomic constraint appear, for example, when the arm's length restricts the position of a robot's hand. Differentiating a holonomic constraint $C(\boldsymbol{q}) = 0$ yields a constraint involving the velocity $G(\boldsymbol{q}, \boldsymbol{v}) = \frac{\partial C}{\partial \boldsymbol{q}}\boldsymbol{v} = 0$, which is simply called a velocity constraint. Hence, each holonomic constraint leads to two first integrals C and G. A Hamiltonian system with holonomic constraints is also a Poisson system; in particular, it is a constrained Hamiltonian system.

A CHNN incorporates the known holonomic constraints $C(\boldsymbol{q})$ and corresponding velocity constraints $G(\boldsymbol{q}, \boldsymbol{v})$ of a Hamiltonian system in the canonical form (Finzi et al., 2020b). The original study suggested that CHNN may learn holonomic constraints from data, but this has not been tested. For modeling a constrained Hamiltonian system, it is sufficient to incorporate only velocity constraints $G(\boldsymbol{q}, \boldsymbol{v})$ because a holonomic constraint $C(\boldsymbol{q})$ is implicitly satisfied if the corresponding velocity constraint $G(\boldsymbol{q}, \boldsymbol{v})$ is satisfied. Celledoni et al. (2022) used such formulation, and extended HNN and CHNN to systems on non-Euclidean spaces. A neural projection method learns fixed holonomic constraints, as well as inequality constraints, which are outside the scope of this study (Yang et al., 2020). This method updates the state by solving an optimization problem similar to Eq. (3) iteratively using the gradient descent method at every training step. Subsequently, it applies the backpropagation algorithm to all the optimization iterations. Thus, it has high computational and memory costs.

These studies mainly focused on physically-induced holonomic constraints and may not work for other first integrals, as shown in Appendices D.3 and D.5. However, the purpose of FINDE is to find and preserve general first integrals, including energy and mass not limited by constraints.

Dirac Structure A Dirac structure is named after a Dirac bracket, a generalization of the Poisson bracket (van der Schaft & Jeltsema, 2014), and can be found in various systems. For a rolling disk, the direction in which the disk can move forward without slipping is limited by the disk's orientation. This constraint is called a non-holonomic constraint. In an electric circuit, when elements are connected in series, the current flow through each element is always the same. This constraint is called Kirchhoff's current law. One can find Dirac structures in these systems. The dissipative SymODEN was proposed to model a port-Hamiltonian system in the canonical form (Zhong et al., 2020b), which is a special case of the Dirac structure. To the best of our knowledge, a neural network model for a general Dirac structure has not yet been proposed. FINDE is the first neural network method to learn Dirac structures better than NODE can, even though it is not specialized for Dirac structures.

PDE with Mass Conservation The total mass of a PDE system is sometimes preserved (Furihata & Matsuo, 2010). The KdV equation is a Hamiltonian system that describes shallow water waves, in which the energy and total mass are preserved. The Cahn–Hilliard equation is a model of phase separation of copolymer melts, in which the total mass is preserved, but the energy is dissipated. In general, a quantity in an area is preserved if its flux entering minus its flux leaving is zero. Deep conservation extracts latent dynamics of a PDE system and preserves a quantity of interest by forcing its flux to be zero (Lee & Carlberg, 2021). HNN++ also ensures mass conservation by designing a coefficient matrix that determines local interaction (Matsubara et al., 2020).

General First Intergals A concurrent study, "Constants-of-motion network," introduced the penalty loss function so that NODEs learn to preserve first integrals (Kasim & Lim, 2022); however, unlike other related methods, this method does not guarantee preservation. A Noether network was proposed to model videos that do not always capture physical phenomena (Alet et al., 2021). A subset of the latent variable is assumed to represent image features that do not change during a video, such as the appearance of objects. For prediction, these features are forced not to change. The Noether network is potentially useful for learning physical phenomena from videos, but is more similar to semantic manipulation of latent variables (Shen et al., 2020).

Some studies have investigated methods that do not predict dynamics but specialize in finding first integrals (Fukunaga & Olsen, 1971; Liu & Tegmark, 2021). These methods can be used to help FINDE determine the hyperparameter K. They commonly estimate the number $(N-K)$ of dimensions of the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$ of the submanifold M' at point $\boldsymbol{u}$ using its neighbors. For example, AI Poincaré proposed by Liu & Tegmark (2021) assumes that all data points share the submanifold $\mathcal{M}'$ and uses an autoencoder to reconstruct the tangent space $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}'$. Hence, it can only process a single long time series with fixed first integrals. In contrast, our proposed FINDE can leverage a dataset of multiple time series with different values of the first integrals.

B Details of Methods

B.1 Derivation of FINDE

Continuous FINDE (cFINDE) Let $\boldsymbol{u}^s$ denote a current state and $\hat{f}$ denote a vector field. After a time interval Δt, the state transitions to $\tilde{\boldsymbol{u}}^{s+1}$. A typical projection method projects the state $\tilde{\boldsymbol{u}}^{s+1}$ onto a submanifold $\mathcal{M}'$ and obtains a state $\boldsymbol{u}^{s+1}$, which preserves the first integrals $\boldsymbol{V} = (V_1\ \dots\ V_K)^\top$. This procedure is defined as an optimization problem in Eq. (3);

$$\arg\min_{\boldsymbol{u}^{s+1}} \|\boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1}\| \text{ subject to } V_k(\boldsymbol{u}^{s+1}) - V_k(\boldsymbol{u}^s) = 0 \text{ for } k = 1, \dots, K. \tag{A1}$$

One can solve the problem using the method of Lagrange multipliers. A Lagrangian function is

$$F(\boldsymbol{u}^{s+1}, \boldsymbol{\lambda}) = \tfrac{1}{2}||\boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1}||_2^2 + (\boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s))^\top \boldsymbol{\lambda}', \tag{A2}$$

where $\boldsymbol{\lambda}'$ is the Lagrange multiplier. The stationary point satisfies

$$\begin{aligned} \tfrac{\partial F}{\partial \boldsymbol{u}^{s+1}} &= \boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1} + \left(\tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^\top \boldsymbol{\lambda}' = \boldsymbol{0}, \\ \tfrac{\partial F}{\partial \boldsymbol{\lambda}'} &= \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) = \boldsymbol{0}. \end{aligned} \tag{A3}$$

Subsequently, a projection method can be redefined as

$$\begin{aligned} \boldsymbol{u}^{s+1} &= \tilde{\boldsymbol{u}}^{s+1} - \left(\tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^\top \boldsymbol{\lambda}', \\ \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) &= \boldsymbol{0}. \end{aligned} \tag{A4}$$

We transform Eq. (A4) into

$$\begin{aligned} \tfrac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^s}{\Delta t} &= \tfrac{\tilde{\boldsymbol{u}}^{s+1} - \boldsymbol{u}^s}{\Delta t} - \left(\tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^\top \boldsymbol{\lambda}, \\ \tfrac{\boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s)}{\Delta t} &= \boldsymbol{0}, \end{aligned} \tag{A5}$$

where $\boldsymbol{\lambda} = \boldsymbol{\lambda}'/\Delta t$. Taking the limit as $\Delta t \to +0$, we obtain Eq. (4);

$$\begin{aligned} f(\boldsymbol{u}^s) &= \hat{f}(\boldsymbol{u}^s) - \left(\tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^s}\right)^\top \boldsymbol{\lambda}, \\ \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{V}(\boldsymbol{u}^s) &= \boldsymbol{0}. \end{aligned} \tag{A6}$$

The second equation ensures that a state transition following the new vector field f preserves the first integrals $\boldsymbol{V}$. By eliminating the Lagrange multiplier $\boldsymbol{\lambda}(\boldsymbol{u})$, we define the cFINDE as in Eq. (6), that is,

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u}) = (I - Y(\boldsymbol{u}))\hat{f}(\boldsymbol{u}) \text{ for } Y(\boldsymbol{u}) = M(\boldsymbol{u})^\top (M(\boldsymbol{u})M(\boldsymbol{u})^\top)^{-1} M(\boldsymbol{u}), \tag{A7}$$

where $M = \frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}}$. Because of the above derivation, the cFINDE can be considered a continuous-time version of a projection method. The preservation of first integrals can be proved as follows.

Proof of Theorem 1.

$$\begin{aligned} \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{V}(\boldsymbol{u}) &= \tfrac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}} \tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} \\ &= M(\boldsymbol{u}) f(\boldsymbol{u}) \\ &= M(\boldsymbol{u})(I - M(\boldsymbol{u})^\top (M(\boldsymbol{u})M(\boldsymbol{u})^\top)^{-1} M(\boldsymbol{u}))\hat{f}(\boldsymbol{u}) \\ &= (M(\boldsymbol{u}) - (M(\boldsymbol{u})M(\boldsymbol{u})^\top)(M(\boldsymbol{u})M(\boldsymbol{u})^\top)^{-1} M(\boldsymbol{u}))\hat{f}(\boldsymbol{u}) \\ &= (M(\boldsymbol{u}) - M(\boldsymbol{u}))\hat{f}(\boldsymbol{u}) \\ &= \boldsymbol{0}. \end{aligned}$$

Hence, it holds that $\frac{\mathrm{d}}{\mathrm{d}t}V_k(\boldsymbol{u}) = 0$ for $k = 1, \dots, K$, indicating that the cFINDE $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ preserves all first integrals V_k in continuous time. □

Discrete FINDE (dFINDE) For dFINDE, we take the discrete gradient of the Lagrangian equation in Eq. (A2) and obtain the discrete version of the necessary conditions for the stationary point;

$$\begin{aligned} \overline{\nabla}_{(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)}F = \boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1} + \overline{M}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)\boldsymbol{\lambda}' = \boldsymbol{0}, \\ \tfrac{\partial F}{\partial \boldsymbol{\lambda}'} = \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) = \boldsymbol{0}. \end{aligned} \tag{A8}$$

$\overline{M}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)$ corresponds to the Jacobian $\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}}$. By substituting the base model $\frac{\tilde{\boldsymbol{u}}^{s+1}-\boldsymbol{u}^s}{\Delta t^s} = \hat{\psi}(\boldsymbol{u}^s;\Delta t^s)$ and the dFINDE $\frac{\boldsymbol{u}^{s+1}-\boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)$ into the above equation and dividing the first equation by Δt, we obtain Eq. (8);

$$\begin{aligned} \tfrac{\boldsymbol{u}^{s+1}-\boldsymbol{u}^s}{\Delta t^s} &= \psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s), \\ \psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s) &= \hat{\psi}(\boldsymbol{u}^s;\Delta t^s) - \overline{M}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)^\top \boldsymbol{\lambda}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s), \\ \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) &= \boldsymbol{0}, \end{aligned} \tag{A9}$$

where $\boldsymbol{\lambda} = \boldsymbol{\lambda}'/\Delta t^s$. By eliminating the Lagrange multiplier $\boldsymbol{\lambda}$, we define the dFINDE as in Eq. (10), that is,

$$\tfrac{\boldsymbol{u}^{s+1}-\boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s) = (I - \overline{Y}(\boldsymbol{u}^{s+1}; \boldsymbol{u}^s))\hat{\psi}(\boldsymbol{u}^s;\Delta t^s) \text{ for } \overline{Y} = \overline{M}^\top(\overline{M}\,\overline{M}^\top)^{-1}\overline{M}. \tag{A10}$$

The preservation of first integrals can be proved as follows.

Proof of Theorem 2.

$$\begin{aligned} \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s) &= \overline{M}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)(\boldsymbol{u}^{s+1} - \boldsymbol{u}^s) \\ &= \overline{M}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s)\psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)\Delta t^s \\ &= \overline{M}(I - \overline{M}^\top(\overline{M}\,\overline{M}^\top)^{-1}\overline{M})\hat{\psi}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)\Delta t^s \\ &= (\overline{M} - (\overline{M}\overline{M}^\top)(\overline{M}\,\overline{M}^\top)^{-1}\overline{M})\hat{\psi}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)\Delta t^s \\ &= (\overline{M} - \overline{M})\hat{\psi}(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)\Delta t^s \\ &= \boldsymbol{0}. \end{aligned}$$

Hence, it holds that $V_k(\boldsymbol{u}^{s+1}) = V_k(\boldsymbol{u}^s)$ for $k = 1,\dots,K$, indicating that the dFINDE $\frac{\boldsymbol{u}^{s+1}-\boldsymbol{u}^s}{\Delta t^s} = \psi(\boldsymbol{u}^{s+1},\boldsymbol{u}^s;\Delta t^s)$ preserves all first integrals V_k in discrete time. □

B.2 Discrete Gradient

A discrete gradient is a discrete analogue to a gradient (Furihata & Matsuo, 2010; Gonzalez, 1996; Hong et al., 2011). Discrete gradients that satisfy Definition 2 are not unique, and many variations have been proposed. For a neural network, Matsubara et al. (2020) proposed the automatic discrete differentiation algorithm (ADDA). We briefly introduce the algorithm in the case of finite-dimensional Euclidean spaces. The differential $\mathrm{d}g$ of a function $g : \mathbb{R}^N \to \mathbb{R}^M$ is a linear operator $\mathrm{d}g_{\boldsymbol{u}} : \mathbb{R}^N \to \mathbb{R}^M$ at point $\boldsymbol{u}$ and satisfies

$$\lim_{||\boldsymbol{h}||_{\mathbb{R}^N}\to 0} \frac{||g(\boldsymbol{u}+\boldsymbol{h}) - g(\boldsymbol{u}) + \mathrm{d}g_{\boldsymbol{u}}(\boldsymbol{h})||_{\mathbb{R}^M}}{||\boldsymbol{h}||_{\mathbb{R}^N}} = 0. \tag{A11}$$

The differential $\mathrm{d}g$ acting on a vector $\boldsymbol{w}$ is equivalent to the product of a vector $\boldsymbol{w}$ with the Jacobian $J_g(\boldsymbol{u})$ of the function g at point $\boldsymbol{u}$: $\mathrm{d}g_{\boldsymbol{u}}(\boldsymbol{w}) = J_{g(\boldsymbol{u})}\boldsymbol{w}$. Similarly, according to the chain rule, the differential $d(h \circ g)$ of a composition $h \circ g$ of functions g, h is equivalent to the multiplication with a series $J_{h(g(\boldsymbol{u}))}J_{g(\boldsymbol{u})}$ of Jacobians. Therefore, the automatic differentiation algorithm obtains the differential of a neural network. The differential $\mathrm{d}g$ of a function $g : \mathbb{R}^N \to \mathbb{R}$ is a horizontal vector, and the gradient ∇g of the function g is a vertical vector dual to the differential. Therefore, the gradient ∇g is obtained by transposing the differential $\mathrm{d}g$. The ADDA replaces each Jacobian with its discrete analogue. For linear layers, such as fully-connected and convolution layers, the discrete Jacobian is identical to the ordinary Jacobian. For element-wise nonlinear layers, such as activation functions, a diagonal matrix composed of the slopes between two inputs can act as the discrete Jacobian. A discrete gradient obtained by the above steps satisfies Definition 2.

B.3 Prediction and Training Procedures

For ODEs modeled by neural networks, various training and prediction strategies have been proposed to date (Chen et al., 2018; 2020; Course et al., 2020; Matsubara et al., 2020; Zhong et al., 2020a); FINDE can adopt any of these. In our experiments, we used the following simple strategies.

In the case of the cFINDE and base models, taking a state $\boldsymbol{u}_{\mathrm{GT}}^s$ from the dataset, a numerical integrator solves the ODE $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ and predicts the next state $\boldsymbol{u}_{\text{pred.}}^{s+1}$. This process can be informally expressed as

$$\boldsymbol{u}_{\text{pred.}}^{s+1} \simeq \boldsymbol{u}_{\mathrm{GT}}^s + \int_{t^s}^{t^s+\Delta t^s} f(\boldsymbol{u}(\tau))\mathrm{d}\tau \text{ for } \boldsymbol{u}(t^s). \tag{A12}$$

We solved this integration using torchdiffeq.odeint. The prediction accuracy can be evaluated using the difference between the predicted state $\boldsymbol{u}_{\text{pred.}}^{s+1}$ and ground truth $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ taken from the dataset. We normalized the difference by the time-step size Δt^s and defined the *1-step error* $\mathcal{L}_{\text{1-step}}$ as

$$\mathcal{L}_{\text{1-step}}(\boldsymbol{u}_{\text{pred.}}^{s+1}; \boldsymbol{u}_{\mathrm{GT}}^{s+1}, \boldsymbol{u}_{\mathrm{GT}}^s, \Delta t^s) = \left\| \frac{\boldsymbol{u}_{\mathrm{GT}}^{s+1} - \boldsymbol{u}_{\mathrm{GT}}^s}{\Delta t^s} - \frac{\boldsymbol{u}_{\text{pred.}}^{s+1} - \boldsymbol{u}_{\mathrm{GT}}^s}{\Delta t^s} \right\|_2^2. \tag{A13}$$

The cFINDE and base models were trained to minimize the 1-step error $\mathcal{L}_{\text{1-step}}$.

In the case of the dFINDE, the next state $\boldsymbol{u}_{\text{pred.}}^{s+1}$ is predicted by solving Eq. (10) as an implicit scheme; in particular,

$$\arg\min_{\boldsymbol{u}_{\text{pred.}}^{s+1}} \left\| \frac{\boldsymbol{u}_{\text{pred.}}^{s+1} - \boldsymbol{u}_{\mathrm{GT}}^s}{\Delta t^s} - (I - \overline{Y}(\boldsymbol{u}_{\text{pred.}}^{s+1}, \boldsymbol{u}_{\mathrm{GT}}^s))\hat{\psi}(\boldsymbol{u}_{\mathrm{GT}}^s; \Delta t^s) \right\|. \tag{A14}$$

Therefore, prediction by the dFINDE is implicit. For evaluation, we solved this scheme using scipy.optimize.fsolve and obtained the 1-step error in Eq. (A13). However, during the training phase, the ground truth $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ of the next state is known. Hence, we substituted this into Eq. (10), and then used the difference between the left- and right-hand sides of the dFINDE as the loss function:

$$\mathcal{L}_{\text{dFINDE}}(\boldsymbol{u}_{\mathrm{GT}}^{s+1}, \boldsymbol{u}_{\mathrm{GT}}^s, \Delta t^s) = \left\| \frac{\boldsymbol{u}_{\mathrm{GT}}^{s+1} - \boldsymbol{u}_{\mathrm{GT}}^s}{\Delta t^s} - (I - \overline{Y}(\boldsymbol{u}_{\mathrm{GT}}^{s+1}, \boldsymbol{u}_{\mathrm{GT}}^s))\hat{\psi}(\boldsymbol{u}_{\mathrm{GT}}^s; \Delta t^s) \right\|_2^2. \tag{A15}$$

The discrete Jacobian $\overline{M}$ (and hence $\overline{Y}$) can be obtained explicitly, and an explicit numerical integrator can be used for the base model $\hat{\psi}$. Hence, the process to obtain the value of the loss function is explicit, and the dFINDE can be trained in an explicit way, whereas the prediction is still implicit.

Some previous studies have proposed alternative strategies. For example, a loss function can be defined as the sum of the errors at multiple time points during a long-term prediction. The cFINDE can naturally adopt such a training strategy, and the dFINDE can adopt it after a minor modification. While it is helpful to pursue absolute performance, it requires additional hyperparameters, such as the length of prediction time, and additional effort to adjust them. We used the 1-step error in the present study for simplicity and fair comparisons.

The function $V(\boldsymbol{u})$ learning a first integral may become a constant function during training; subsequently, its Jacobian matrix vanishes ($\frac{\partial V(\boldsymbol{u})}{\partial \boldsymbol{u}} \equiv 0$). In this case, our algorithm returns a division-by-zero error because it requires the inverse of the matrix $\frac{\partial V(\boldsymbol{u})}{\partial \boldsymbol{u}} \frac{\partial V(\boldsymbol{u})}{\partial \boldsymbol{u}}^\top$ for the projection. We have not taken any special measures to prevent such errors, but no errors occurred in any experiments with proper settings. The division-by-zero errors have occurred only when FINDE assumes an unreasonable number of first integrals (e.g., $K = 6$ for the double pendulum, which has five first integrals). FINDE works correctly even when the functions $f(\boldsymbol{u})$ and $V(\boldsymbol{u})$ learn the same first integrals; we verified such a case in Section 4.2, where both functions are known.

FINDE learns first integrals point-by-point, and the found first integral is not always consistent over the domain. The same can be said about the energy function of HNN, and this type of problem is an open problem for neural network models of dynamical systems.

C Details of Datasets

To generate each dataset, we used scipy package and the Dormand–Prince method (dopri5) with the default relative tolerance of 10^{-9}, unless otherwise stated. Experiments on the KdV dataset were performed with double precision, and all other experiments were performed with single precision.

Hamiltonian System in Canonical Form: Two-Body Problem A gravitational two-body problem on a 2-dimensional configuration space has a state $\boldsymbol{u}$ composed of the 4-dimensional position $\boldsymbol{q} = (x_1\ y_1\ x_2\ y_2)^\top$ and 4-dimensional velocity $\boldsymbol{v} = (v_{x1}\ v_{y1}\ v_{x2}\ v_{y2})^\top$. This is a second-order ODE, indicating that $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q} = \boldsymbol{v}$. The momentum p_{x_1} of x_1 equals $m_1 v_{x_1}$. The time-derivative $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{v}$ of the velocity $\boldsymbol{v}$ is called the acceleration. The acceleration of x_1 is given by $\frac{\mathrm{d}}{\mathrm{d}t}v_{x_1} = -Gm_1m_2\frac{x_1-x_2}{((x_1-x_2)^2+(y_1-y_2)^2)^{3/2}}$, where G, m_1, and m_2 denote the constant of gravity and masses of two bodies, respectively. The same process applies for the remaining positions.

The total energy of the two-body problem is given by

$$H = \frac{1}{2}(m_1(v_{x1}^2 + v_{y1}^2) + m_2(v_{x2}^2 + v_{y2}^2)) - \frac{Gm_1m_2}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}}. \tag{A16}$$

The first and second terms denote the kinetic and potential energies, respectively. The two-body problem is a Hamiltonian system, and the dynamics mentioned above can be rewritten as Hamilton's equation. The Hamiltonian H is one of the first integrals; the two-body problem has other first integrals, such as the linear momenta in the x- and y-directions

$$p_x = \frac{m_1 v_{x1} + m_2 v_{x2}}{m_1 + m_2},\ p_y = \frac{m_1 v_{y1} + m_2 v_{y2}}{m_1 + m_2}, \tag{A17}$$

and angular momentum (Hairer et al., 2006).

We set G, m_1, and m_2 to 1.0. The initial distance $r_1 = \sqrt{x_1^2 + y_1^2}$ of a mass m_1 from the origin was set to $r_1 \sim \mathcal{U}(0.5, 1.0)$, and the initial angle $\theta_1 = \tan^{-1}(\frac{y_1}{x_1})$ was set to $\theta_1 \sim \mathcal{U}(0, 2\pi)$. The initial speed $|v_1| = \sqrt{v_{x_1}^2 + v_{y_1}^2}$ was set to $\frac{1}{2r^2}\epsilon_v$, where $\epsilon_v \sim \mathcal{N}(1, 0.05)$. The initial angle of the velocity was set to $\theta \pm 0.5\pi + \epsilon_\theta\pi$, where $\epsilon_\theta \sim \mathcal{N}(0, 0.05)$. The initial condition of the other mass m_2 was set to the opposite of the mass m_1. Subsequently, the two masses trace elliptical orbits, and when $\epsilon_v = \epsilon_\theta = 0$, they trace exactly circular orbits. In addition, we added a perturbation following $\mathcal{N}(0, 0.01)$ to the velocities of both masses, which corresponds to the center-of-gravity velocity.

We set the time-step size Δt to 0.01 and generated 1,000 time-series of $S = 500$ steps for training and 10 time-series of $S = 10,000$ steps for evaluation. We trained each model for 100,000 iterations.

Hamiltonian System in Non-Canonical Form: KdV equation The KdV equation is a model of shallow water waves and is known to have soliton solutions (Furihata, 2001). The dynamics is given by

$$u_t = -\alpha u u_x + \beta u_{xxx}, \tag{A18}$$

where x denotes the spatial position and the subscripts denote partial derivatives; for example, $u_t = \frac{\partial u}{\partial t}$. The Hamiltonian is given by

$$H(u) = \int -\frac{1}{6}\alpha u^3 - \frac{1}{2}\beta u_x^2\,\mathrm{d}x. \tag{A19}$$

As Hamilton's equation $\frac{\mathrm{d}}{\mathrm{d}t}u = S\nabla H$, the partial differential operator $\frac{\partial}{\partial x}$ acts as the coefficient matrix S. This system is Liouville integrable and has infinitely many first integrals, including the Hamiltonian H, total mass $I_1 = \int u\mathrm{d}x$, and $T_2 = \int u^2\mathrm{d}x$ (Miura et al., 1968). Other first integrals are defined using higher-order partial derivatives.

For PDEs, PINNs are known to provide solutions when symbolic equations and boundary conditions are given (Raissi et al., 2019). We, in contrast, consider learning spatially discretized PDEs as ODEs from observed data and solving them using numerical integrators, in the same context as NODEs and HNNs; this topic has also been studied extensively (Long et al., 2018; Matsubara et al., 2020; Sun et al., 2020; Holl et al., 2020). Following the experiments in a previous study (Matsubara et al., 2020), we discretized the KdV equation in space; it no longer has infinitely many first integrals. We set $\alpha = -6$, $\beta = 1$, spatial size to 10 space units, and space mesh size to 0.2; the system state $\boldsymbol{u}$ had 50 elements. We generated two solitons as the initial condition; each was expressed as $-\frac{12}{\alpha}\kappa^2 \mathrm{sech}^2(\kappa(x-d))$, where the size κ followed $\mathcal{U}(0.5, 2)$ and the initial position d of one soliton was set to be at least 2.0 from that of the other.

We set the time-step size Δt to 0.001 and generated 1,000 time-series of $S = 500$ steps for training and 10 time-series of $S = 10,000$ steps for evaluation, using the discrete gradient method to ensure energy conservation (Furihata, 2001). We trained each model for 30,000 iterations.

Due to the spatial discretization, the KdV dataset contains spatial truncation errors. When the neural network learns this dataset, no spatial truncation errors are additionally introduced. An evaluation using the analytical solution as a dataset or datasets created with different spatial resolutions is included in future work.

Poisson System: Double Pendulum A double pendulum (2-pend) is depicted in Fig. A1. In polar coordinates, this is a Hamiltonian system. The state is composed of the angles (θ_1, θ_2) of the two rods and their angular velocities (ω_1, ω_2). This is also a second-order ODE, indicating that $\frac{\mathrm{d}}{\mathrm{d}t}\theta_1 = \omega_1$ and $\frac{\mathrm{d}}{\mathrm{d}t}\theta_2 = \omega_2$.

Let l_1, l_2 denote the lengths of the two rods, m_1, m_2 denote the masses of the two weights, and g denote the gravitational acceleration. The acceleration is given by

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\omega_1 &= \frac{m_2 g \sin\theta_2 \cos\Delta - (l_1\omega_1^2 \cos\Delta + l_2\omega_2^2) m_2 \sin\Delta - (m_1+m_2) g \sin\theta_1}{l_1(m_1 + m_2 \sin^2\Delta)}, \\
\frac{\mathrm{d}}{\mathrm{d}t}\omega_2 &= \frac{(m_1+m_2)(l_1\omega_1^2 \sin\Delta - g\sin\theta_2 + g\sin\theta_1\cos\Delta) + m_2 l_2 \omega_2^2 \sin\Delta\cos\Delta}{l_2(m_1 + m_2\sin^2\Delta)},
\end{aligned} \tag{A20}$$

where $\Delta = \theta_1 - \theta_2$. In 2-dimensional Cartesian coordinates, the state is composed of the positions (x_1, y_1, x_2, y_2) of the two masses and the corresponding velocities $(v_{x1}, v_{y1}, v_{x2}, v_{y2})$. The position is transformed by $x_1 = l_1 \sin\theta_1$, $y_1 = l_1\cos\theta_1$, $x_2 = x_1 + l_2\sin\theta_2$, and $y_2 = y_1 + l_2\cos\theta_2$, and the velocity is transformed accordingly. The total energy H is given by

$$H = \frac{1}{2}(m_1(v_{x_1}^2 + v_{y_1}^2) + m_2(v_{x_2}^2 + v_{y_2}^2)) + g(m_1 y_1 + m_2 y_2). \tag{A21}$$

The first and second terms denote the kinetic and potential energies, respectively. The double pendulum is no longer a Hamiltonian system in Cartesian coordinates. Because the lengths of the two rods are constant, the double pendulum has two constraints on the position: $l_1^2 = x_1^2 + y_1^2$ and $l_2^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$. These constraints are holonomic constraints, and they lead to constraints involving the velocity, namely $0 = x_1 v_{x_1} + y_1 v_{y_1}$ and $0 = (x_2 - x_1)(v_{x_2} - v_{x_1}) + (y_2 - y_1)(v_{y_2} - v_{y_1})$. When the constraints involving the velocity are satisfied, the holonomic constraints are implicitly satisfied. Therefore, the number of first integrals is five; however, three first integrals are sufficient to determine the dynamics. The dynamics is degenerate and classified as a constrained Hamiltonian system, or a Poisson system in a more general case.

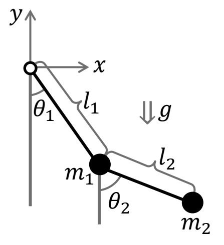

Figure A1: Diagram of the double pendulum.

We set the masses of the two weights to $m_1 = m_2 = 1.0$ and the gravitational acceleration g to 9.8. We set the lengths l_1, l_2 of the two rods to follow $\mathcal{U}(0.9, 1.1)$, the initial angles θ_1, θ_2 to follow $\mathcal{U}(-0.5, 0.5)$, and the initial angular velocities $\dot\theta_1, \dot\theta_2$ to follow $\mathcal{U}(-0.1, 0.1)$.

We set the time-step size Δt to 0.1 and generated 1,000 time-series of $S = 500$ steps for training and 10 time-series of $S = 5,000$ steps for evaluation. We trained each model for 100,000 iterations.

Dirac Structure: FitzHugh–Nagumo Model R. FitzHugh proposed a model of the electrical dynamics of a biological neuron, and J. Nagumo created an equivalent electric circuit. This model is called the FitzHugh–Nagumo model (Izhikevich & FitzHugh, 2006) and is a modified version of the van der Pol oscillator; the state oscillates when the magnitude of the external current source I is within an appropriate range. The circuit comprises a resistor R, inductor L, capacitor C, tunnel diode D, and voltage source E connected as shown in Fig. A2. The whole circuit is connected to an external current source I. Let I_R denote the current through the resistor R, and V_R denote the applied voltage. Ohm's law and other properties of the elements give $V_R = I_R R$, $C\frac{\mathrm{d}}{\mathrm{d}t}V_C = I_C$, $L\frac{\mathrm{d}}{\mathrm{d}t}I_L = V_L$, and $I_D = D(V_D)$, where we treat D as a nonlinear function. Kirchhoff's current law (KCL) obtains $I_C + I_D + I_R = I$ and $I_R = I_L$, and Kirchhoff's voltage law (KVL) obtains $V_C = V_D = V_R + V_L + E$. We denote $W = I_R$ and $V = V_C$, and set $L = 1/0.08$, $R = 0.8$, $C = 1.0$, $V_E = -0.7$, and $D(V) = V^3/3 - V$. Subsequently, we obtain the FitzHugh–Nagumo model of the original parameters as

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}V &= V - V^3/3 - W + I,\\ \frac{\mathrm{d}}{\mathrm{d}t}W &= 0.08(V + 0.7 - 0.8W).\end{aligned} \tag{A22}$$

Due to the resistor R, the FitzHugh–Nagumo model is not an energy-conserving system.

Consider a situation where the current through and the voltage applied to stateful elements (capacitors and inductors) are measurable, but the connections between the elements are unknown. We treated I_C, I_L, V_C, V_L as the system state $\boldsymbol{u}$. Because the state is in 4-dimensional space and the dynamics is intrinsically 2-dimensional, there exist two first integrals; for example, but not limited to, $I = I_C + D(V_C) + I_L$ and $E = V_C - I_L R - V_L$. This type of electric circuit is an example of a Dirac structure because the state variables are constrained by the circuit topology and Kirchhoff's current and voltage laws (van der Schaft & Jeltsema, 2014). From the viewpoint of generalized Hamiltonian systems, (I_L, V_C) corresponds to the position, and (V_L, I_C) corresponds to the momentum. The electric circuit can be described as a port-Hamiltonian system in a non-canonical form. Because of the non-canonical form, the FitzHugh–Nagumo model is outside the scope of CHNN and dissipative SymODEN (Finzi et al., 2020b; Zhong et al., 2020b).

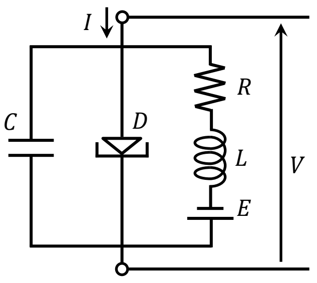

Figure A2: Circuit diagram of FitzHugh–Nagumo model (Izhikevich & FitzHugh, 2006).

We set the external current source I to follow $\mathcal{U}(0.7, 1.1)$, set the initial values of V and W to follow $\mathcal{U}(-1.5, 1.5)$ and $\mathcal{U}(0.0, 2.0)$, and transformed them to the state.

We set the time-step size Δt to 0.1 and generated 1,000 time-series of $S = 500$ steps for training and 10 time-series of $S = 2,000$ steps for evaluation. We trained each model for 30,000 iterations.

D Additional Results and Discussion

D.1 Demonstration of First Integral Preservation

In Fig. 1, we examined a mass-spring system and FINDE using the leapfrog integrator. We also examined the case with the Dormand–Prince integrator (dopri5), as shown in Fig. A3. We increased the number of steps to 10^5, and displayed the MSEs of the state instead of the state itself. First, we focus on the energy. Even using the Dormand–Prince integrator, a fourth-order method, the energy is slightly decreased. The cFINDE with the Dormand–Prince integrator shows the same tendency. This phenomenon is due to temporal discretization errors and is called energy drift. The dFINDE with the Dormand–Prince integrator significantly suppresses the error in energy. The remaining error is caused by rounding errors.

When the focus is on the MSEs of the state, the trend is different: the dFINDE with the Dormand–Prince integrator suffers from the most significant errors in state. Although the dFINDE is designed to eliminate temporal discretization errors in energy, it does not necessarily reduce those in state. In contrast, the Dormand–Prince integrator is designed to suppress temporal discretization errors in state.

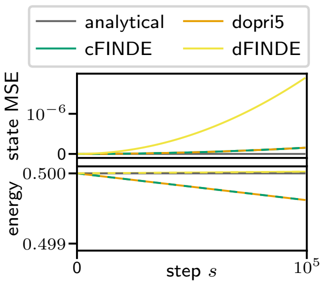

Figure A3: Integration of a known mass-spring system by Dormand–Prince integrator. (top) Mean squared errors in states predicted by comparison methods. (bottom) Energy calculated from the states predicted.

Therefore, there is no guarantee that the dFINDE improves the prediction performance when defined using errors in state. Conversely, the experimental results in Table 3 demonstrate that the dFINDE is superior to the base model and cFINDE in VPT. This is because dFINDE reduces the modeling errors rather. For the mass-spring system, the governing equation is already known as an ODE and is discretized by the dFINDE, leading to temporal discretization errors. However, when dFINDE learns dynamics from data, the training data points are already sampled in discrete time, and the dFINDE predicts future states in discrete time. Therefore, no temporal discretization error occurs, and we obtain only the advantages of exactly preserving the first integral.

This type of paradox has been repeatedly discovered in previous studies. For example, the leapfrog integrator and discrete gradient method are second-order methods. However, they are superior to the Dormand–Prince integrator when combined with neural networks and learning dynamics from data (Matsubara et al., 2020). For better learning (i.e., smaller modeling errors), the preservation of specific properties of target systems is more important than the order of accuracy.

D.2 Symbolic Regression of Found First Integrals

Using gplearn (based on genetic programming), we performed a symbolic regression of the first integrals $\boldsymbol{V}$ found by the neural network. We prepared addition, subtraction, multiplication, and division as candidate operations, used Pearson's correlation coefficient as the evaluation criterion, set the early stopping threshold to 0.9, and set the population size to 10,000. We set the other hyperparameters to their default values, e.g., the maximum number of generations was 20.

We summarize the regression results of the HNN with cFINDE for $K = 2$ trained using the two-body dataset in Table A1. Note that Pearson's correlation coefficient is invariant to biases and scale factors. FINDE is also invariant because it only uses the directions of the gradients of first integrals. Hence, we removed biases and scale factors from the regression results. When the focus is on the symbolic regression of the training data, V_1, V_1, V_2, and V_2 for trials 0, 1, 2, and 3 are identical to the linear momentum in the x-direction up to scale factors; recall that we set $m_1 = m_2 = 1.0$ and see Eq. (A17). V_2, V_2, V_1, and V_1 for trials 0, 1, 2, and 3 are also identical to the linear momentum in the y-direction. V_1 and V_2 for trial 4 are weighted sums of the linear momenta in the x- and y-directions; in particular, they can be regarded as the linear momenta in the $(1,-1)$- and $(1,1)$-directions, respectively.

When the quantities $V_1(\boldsymbol{u})$ and $V_2(\boldsymbol{u})$ are first integrals, any function of only $V_1(\boldsymbol{u})$, $V_2(\boldsymbol{u})$, and arbitrary constants is a first integral functionally dependent on $V_1(\boldsymbol{u})$ and $V_2(\boldsymbol{u})$. Thus, it is in principle impossible to re-discover a first integral as a well-known symbolic expression, and a failure in symbolic regression is not a problem in any way. Previous studies introduced certain constraints (such as "gauge fixing") for symbolic regression (Liu & Tegmark, 2021); a combination of such method may improve the results. However, recent studies on neural networks have revealed that typical initialization and training procedures tend to learn simple functions (Barrett & Dherin, 2021; Cao et al., 2021). Additionally, the symbolic regression limited the depth of the computation graph, biasing the results toward simple functions; hence, the found first integrals were identical to the well-known forms and were separated in the x- and y-directions in most cases.

The same is true for the symbolic regression of the test data, except for V_1 for trial 0, which had a small perturbation α. Because of the limited extrapolation ability, neural networks cannot always accurately represent functions outside the training data range. Once first integrals are found by FINDE and identified as equations by symbolic regression, one can use the equations instead of neural networks, ensuring the preservation of first integrals in the entire domain. From these results, we can conclude that cFINDE identified the linear momenta.

The state of the KdV dataset has 50 elements, which is too large to apply a symbolic regression. For the 2-pend and FitzHugh–Nagumo datasets, we did not find consistent equations of first integrals. For example, the symbolic regression identified a quantity $x_1^2 - y_1$ as a first integral in the 2-pend dataset, which is not directly related to well-known first integrals. When the angle θ_1 of the upper rod is small, y_1 takes a value close to -1, and the quantity $x_1^2 - y_1$ is close to $x_1^2 + y_1^2$, which is a well-known first integral, namely the square l_1^2 of the upper rod length l_1. It is difficult to determine whether this inaccuracy is because of the training of FINDE or symbolic regression. There may still be room for improvement in the training of FINDE or symbolic regression.

Table A1: Symbolic Regression of First Integrals Found in Two-Body Problem

	Training Data		**Test Data**	
Trial	V_1	V_2	V_1	V_2
0	$v_{x1}+v_{x2}$	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}+\alpha$	$v_{y1}+v_{y2}$
1	$v_{x1}+v_{x2}$	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}$	$v_{y1}+v_{y2}$
2	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}$	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}$
3	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}$	$v_{y1}+v_{y2}$	$v_{x1}+v_{x2}$
4	$v_{x1}+v_{x2}-v_{y1}-v_{y2}$	$v_{x1}+v_{x2}+v_{y1}+v_{y2}$	$v_{x1}+v_{x2}-v_{y1}-v_{y2}$	$v_{x1}+v_{x2}+v_{y1}+v_{y2}$

We removed biases and scale factors. $\alpha = 0.003(y_1+y_2)(v_{x2}+x_1+y_1(v_{x2}+y_1+y_2)+1.402)$.

Table A2: Results with Known Holonomic Constraints.

	2-pend		**2-body**	
Model	**1-step**$\downarrow$	**VPT**$\uparrow$	**1-step**$\downarrow$	**VPT**$\uparrow$
NODE	0.82 $\pm$0.02	0.110 $\pm$0.035	144.21 $\pm$12.65	0.134 $\pm$0.014
HNN (Greydanus et al., 2019)	6220.26 $\pm$91.57	0.002 $\pm$0.000	5.17 $\pm$0.57	0.362 $\pm$0.026
CHNN (Finzi et al., 2020b)	0.07 $\pm$0.00	0.928 $\pm$0.036	(not working)	
NODE+cFINDE	0.71 $\pm$0.04	0.461 $\pm$0.071	163.64 $\pm$9.79	0.147 $\pm$0.024
HNN+cFINDE	236.51 $\pm$7.15	0.020 $\pm$0.002	8.32 $\pm$0.43	0.476 $\pm$0.040

D.3 Comparison with Model of Known Holonomic Constraints

The double pendulum (2-pend) is classified as a constrained Hamiltonian system. CHNN was proposed for cases when holonomic constraints are known (Finzi et al., 2020b). We evaluated comparison methods under the assumption that the holonomic constraints were known. We summarized the results in Table A2. The HNN, without constraints, completely failed to learn the dynamics. This is unsurprising because the dynamics of the double pendulum is outside the scope of the HNN. The two known holonomic constraints lead to two constraints involving the velocity; the CHNN took into account all four known constraints and worked remarkably. The HNN with cFINDE was given all four known constraints as the first integrals, but did not work properly. The original purpose of projection methods is to eliminate temporal discretization errors of first integrals but not to change the class to which the dynamics belong. Therefore, when a target system is not a subject of the base model, the base model with FINDE does not work. The NODE learns an ODE in a general way, and thus constrained Hamiltonian systems are included in its subjects. Given all four known constraints, the NODE with cFINDE worked better but never surpassed the CHNN.

However, the CHNN works only for Hamiltonian systems in the canonical form with holonomic constraints. We also evaluated comparison methods using the 2-body dataset under the assumption that the linear momenta were known as first integrals. The CHNN attempted to obtain the inverse of a singular matrix and could not learn the dynamics. In contrast, the cFINDE improved the performances of both NODE and HNN.

Existing methods (e.g., HNN and CHNN) assume geometric structures (e.g., Hamiltonian structure) described in Appendix A in order to guarantee conservation laws. When multiple structures are assumed at the same time, they must be integrated using appropriate prior knowledge. If it is possible, it would achieve extremely high performance. Otherwise, the geometric structures would conflict with each other and would not produce an appropriate model. This is the reason why CHNN failed to learn the 2-body dataset and HNN+FINDE failed to learn the 2-pend dataset. In contrast, NODE+FINDE does not assume any geometric structure and assumes first integrals in the most general way, being available to any situation. Hence, FINDE can assume one or more first integrals without changing anything.

When the detailed properties of target systems are known, one can choose the best models. If the chosen model is inappropriate, the training procedure totally fails. FINDE provides a better alternative when prior knowledge is limited. Moreover, a constrained Hamiltonian system can have first integrals other than holonomic constraints and the Hamiltonian. In this case, the CHNN with FINDE is potentially the best choice.

Table A3: Results of NODE with cFINDE on Training Set of 2-Pend Dataset.

		2-pend	
Model	K	**1-step**↓	**VPT**↑
NODE	–	0.76 ±0.02	0.966 ±0.007
+ cFINDE	1	0.72 ±0.06	0.974 ±0.004
	2	0.69 ±0.08	0.981 ±0.014
	3	0.63 ±0.02	0.994 ±0.002
	4	0.67 ±0.05	0.990 ±0.005
	5	0.65 ±0.02	0.998 ±0.000
	6	9.93 ±0.00	0.126 ±0.000

D.4 Reason for High Performance and How to Determine Number of First Integrals

The theoretical explanation for the high performance of neural networks (e.g., HNN) that assume first integrals for physical phenomena is an open question. Sannai et al. (2021) has theoretically shown that neural networks (e.g., CNNs and GNNs) with symmetry have faster learning convergence, and we consider this approach can be applied to the above question. At least for cFINDE and dFinde, we have an intuitive but not rigorous explanation; assuming one more first integral (i.e., increasing K by 1) reduces the number of degrees of freedom in the dynamics by 1, narrows the hypothesis space, accelerates learning convergence, and suppresses generalization errors.

As shown in Table 3, the performance of cFINDE and dFINDE is sensitive to the assumed number K of first integrals. Because K is a hyperparameter, it is basically a subject to be adjusted through evaluations on a validation set. With inappropriately large K, both cFINDE and dFINDE dropped their performance significantly. See the results of the 2-pend and FitzHugh–Nagumo datasets for $K = 6$ and $K = 3$, respectively.

However, the performance drop can be found even with the training set. Table A3 summarizes the prediction performance on the training set of the 2-pend dataset. As was the case with the test set, the performance significantly dropped at $K = 6$. This is because NODE with cFINDE for $K = 6$ assumes the submanifold $\mathcal{M}'$ to be 2-dimensional. The submanifold $\mathcal{M}'$ is in fact 3-dimensional, so NODE with cFINDE for $K = 6$ is incapable of learning the dynamics and performs poorly even on the training set. Hence, the training set is enough to avoid a fatally inappropriate K.

Alternatively, K can be determined by using other methods (e.g., Fukunaga & Olsen (1971); Liu & Tegmark (2021)). Although these methods have some drawbacks introduced in Appendix A, they may be complementary to FINDE.

D.5 Comparison with Modified Neural Projection Method

The neural projection method (NPM) also employs a projection method (Yang et al., 2020). Using a manner similar to Newton's method, it enforces the constraint $C(\boldsymbol{u}) = 0$ by the projection of the state $\boldsymbol{u}$ under the assumption that the quantity $C(\boldsymbol{u})$ is always zero. This assumption holds for some cases (e.g., holonomic constraints in a fixed environment), but not for most first integrals, whose values depend on initial conditions.

For example, the linear momentum in the x-direction of the two-body problem is the first integral expressed as $V(\boldsymbol{u}) = m_1 v_{x1}(t) + m_2 v_{x2}(t)$. This quantity V is constant within a trial (i.e., $V(\boldsymbol{u}(t)) = V(\boldsymbol{u}(0))$) and varies between trials depending on the initial speed $v_{x1}(0)$ and $v_{x2}(0)$. The total energy, the total mass, and many other first integrals depend on the initial condition in the same manner; hence, they are outside the scope of the NPM. In contrast, by imposing the constraint on the gradient $\nabla V = 0$ or discrete gradient $\overline{\nabla} V = 0$, our proposed FINDE keeps the quantity V constant and can handle any first integrals.

For comparison, we replaced the constraint $C(\boldsymbol{u}) = 0$ with $C(\boldsymbol{u}^{s+1}, \boldsymbol{u}^s) = V(\boldsymbol{u}^{s+1}) - V(\boldsymbol{u}^s) = 0$ and adopted the NPM to first integrals varying from trial to trial. We evaluated the modified NPM using the 2-pend dataset. Because the modified NPM is a discrete-time projection method,

Table A4: Comparison with Neural Projection Method (NPM)

K	**dFINDE (proposed)**		**modified NPM**		
	1-step↓	**VPT↑**	**1-step↓**	**VPT↑**	**successful**
1	0.75 ±0.10	0.152 ±0.017	0.73 ±0.08	0.150 ±0.014	5/5
2	0.74 ±0.05	0.271 ±0.111	—	—	0/5
3	0.69 ±0.05	0.447 ±0.081	(0.69 ±0.00)	(0.138 ±0.000)	1/5
4	0.71 ±0.03	0.454 ±0.060	(0.72 ±0.03)	(0.383 ±0.023)	3/5
5	0.86 ±0.09	0.591 ±0.087	0.85 ±0.11	0.364 ±0.134	5/5
6	58.88 ±22.98	0.037 ±0.039	(1.29 ±0.20)	(0.103 ±0.016)	3/5

we compared it with the discrete-time version of the proposed FINDE (dFINDE). The results are summarized in Table A4.

The dFINDE successfully learned the dynamics in all trials, but the modified NPM failed to learn the dynamics in half the trials (see the rightmost column for the numbers of successful trials out of 5). The modified NPM often encountered of the underflow of the time-step size or a division by the zero gradient of the first integral. Even when the learning was successful, the performance of the NPM was inferior to that of the dFINDE. The modified NPM solved the optimization problem in Eq. (3) at every step, but it sometimes diverged or failed to converge, especially in the early phase of learning. The NPM was successful for fixed environments but might be unsuited for general first integrals varying from trial to trial. However, the dFINDE does not require solving an optimization problem during training, making the learning process robust against randomness such as initialization.