Appendix of FINDE: Neural Differential Equations for Finding and Preserving Invariant Quantities

A Hamiltonian System, Its Generalization, and First Integrals

Preliminary In this section, we briefly introduce potential target systems and related works. See, for example, [27, 53] for more details.

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 $\mathcal{T}_{\boldsymbol{u}}\mathcal{M}$. Thus, the NODE defines an ODE [8]. 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 multiplication 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, defined on Darboux coordinates, on which the state $\boldsymbol{u}$ is the pair of generalized position $\boldsymbol{q}$ and generalized momentum $\boldsymbol{p}$. The corresponding coefficient matrix is $S = \left(\begin{smallmatrix} 0 & I_n \\ -I_n & 0 \end{smallmatrix}\right)$ for $2n = N$. The HNN was developed to model Hamiltonian systems in the canonical forms [26].

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 [9]. The KdV equation is also a Hamiltonian system not on Darboux coordinates. For Hamiltonian PDE systems, HNN++ was proposed [38]. According to Darboux's theorem, any Hamiltonian systems 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 implemented such symmetries in their architectures [19, 21]. A pendulum is not symmetric to translation and rotation in space and does not conserve linear and angular momenta (but exchanges 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 $S = \left(\begin{smallmatrix} 0 & I_k & 0 \\ -I_k & 0 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right)$ for $2k < N$. The last $N - 2k$ elements remain unchanged and correspond to 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 the 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 [31].

Constrained Hamiltonian System A constraint $V(\boldsymbol{q}) = 0$ on the position $\boldsymbol{q}$ is called a holonomic constraint and appears, for example, when the position of the hand of a robot is restricted by the

length of the arm. Differentiating a holonomic constraint $V(\boldsymbol{q}) = 0$ yields a constraint involving the velocity $G(\boldsymbol{q}, \boldsymbol{v}) = \frac{\partial V}{\partial \boldsymbol{q}} \boldsymbol{v} = 0$, which is simply called a velocity constraint. Hence, each holonomic constraint leads to two first integrals V and G. A Hamiltonian system with holonomic constraints is also a Poisson system, but in particular, it is a constrained Hamiltonian system.

A CHNN incorporates known holonomic constraints $V(\boldsymbol{q})$ and corresponding velocity constraints $G(\boldsymbol{q}, \boldsymbol{v})$ for a Hamiltonian system in the canonical form [20]. The original study suggested that CHNN may learn holonomic constraints from data but has not tested it. For modeling a constrained Hamiltonian system, it is sufficient to incorporate only velocity constraints $G(\boldsymbol{q}, \boldsymbol{v})$ because a holonomic constraint $V(\boldsymbol{q})$ is implicitly satisfied if the corresponding velocity constraint $G(\boldsymbol{q}, \boldsymbol{v})$ is satisfied. Celledoni et al. [59] used such formulation and extended HNN and CHNN to systems on non-Euclidean spaces. A neural projection method learns holonomic constraints (as well as inequality constraints, which are outside the scope of this study) [62]. This method updates the state by solving an optimization problem similar to Eq. (3) iteratively by gradient descent method at every training step, and then apply backpropagation algorithm to all the optimization iterations. Thus, it consumes much computational cost and memory.

These studies focused mainly on physically-induced holonomic constraints and may not work for other first integrals, as shown in Section D.2 in Appendix. On the other hand, the purpose of FINDE is to find and preserve general first integrals, including energy and mass, not limited to constraints.

A Noether network was proposed to model videos that do not always capture physical phenomena [58]. 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, it is forced not to change. The Noether network is potentially useful to learn physical phenomena from videos, but it is more like semantic manipulation of latent variables [61].

Dirac Structure A Dirac structure is named after a Dirac bracket, which is a generalization of the Poisson bracket [53]. A Dirac structure can be found in various systems. For a rolling disk, the direction in which the disk can move forward without slipping is restricted by the angle at which the disk is facing. This constraint is called a non-holonomic constraint. In an electric circuit, when elements are connected in series, the amount of current flowing 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 [57], which is a special case of Dirac structure. To our best knowledge, a neural network model for a general Dirac structure has not yet been proposed. FINDE is the first neural network method that demonstrates to learn Dirac structures better than NODE, but it is not specialized for Dirac structures.

PDE with Mass Conservation The total mass of a PDE system is sometimes preserved [23]. The KdV equation is a Hamiltonian system and 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 the area is zero. Deep conservation extracts latent dynamics of a PDE system and preserves a quantity of interest by forcing its flux to be zero [34]. HNN++ also ensures the mass conservation by designing a coefficient matrix that determines local interaction [38].

B Details of Methods

B.1 Derivation of Projection Method and FINDE

Let $\boldsymbol{u}^s$ denote a current state and $\hat{f}$ denote a vector field. After a time interval Δt, the state transitions to $\hat{\boldsymbol{u}}^{s+1}$. 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 solving an optimization problem

$$\min ||\boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1}|| \text{ subject to } V_k(\boldsymbol{u}^{s+1}) - V_k(\boldsymbol{u}^s) = \boldsymbol{0} \text{ for } k = 1, \dots, K. \tag{13}$$

One can solve the problem using, for instance but not limited to, the method of Lagrange multipliers. A Lagrangian function is

$$F(\boldsymbol{u}^{s+1}, \boldsymbol{\lambda}) = \tfrac{1}{2}||\boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1}||_2^2 + (\boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^s))^\top \boldsymbol{\lambda}', \tag{14}$$

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

$$\begin{aligned} \frac{\partial F}{\partial \boldsymbol{u}^{s+1}} &= \boldsymbol{u}^{s+1} - \tilde{\boldsymbol{u}}^{s+1} + \left(\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^{\top} \boldsymbol{\lambda}' = \boldsymbol{0}, \\ \frac{\partial F}{\partial \boldsymbol{\lambda}'} &= \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^{s}) = \boldsymbol{0}. \end{aligned} \tag{15}$$

Then, a projection method can be redefined as

$$\begin{aligned} \boldsymbol{u}^{s+1} &= \tilde{\boldsymbol{u}}^{s+1} - \left(\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^{\top} \boldsymbol{\lambda}', \\ \boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^{s}) &= \boldsymbol{0}. \end{aligned} \tag{16}$$

For obtaining FINDE, we transform the above equations into

$$\begin{aligned} \frac{\boldsymbol{u}^{s+1} - \boldsymbol{u}^{s}}{\Delta t} &= \frac{\tilde{\boldsymbol{u}}^{s+1} - \boldsymbol{u}^{s}}{\Delta t} - \left(\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s+1}}\right)^{\top} \boldsymbol{\lambda}, \\ \frac{\boldsymbol{V}(\boldsymbol{u}^{s+1}) - \boldsymbol{V}(\boldsymbol{u}^{s})}{\Delta t} &= \boldsymbol{0}, \end{aligned} \tag{17}$$

where $\boldsymbol{\lambda} = \boldsymbol{\lambda}'/\Delta t$. Alternatively, one can define another Lagrangian function and obtain Eq. (17). Anyway, taking the limit $\Delta t \to +0$, we obtain the cFINDE;

$$\begin{aligned} f(\boldsymbol{u}^{s}) &= \hat{f}(\boldsymbol{u}^{s}) - \left(\frac{\partial \boldsymbol{V}}{\partial \boldsymbol{u}^{s}}\right)^{\top} \boldsymbol{\lambda}, \\ \frac{\mathrm{d}}{\mathrm{d}t} \boldsymbol{V}(\boldsymbol{u}^{s}) &= \boldsymbol{0}. \end{aligned} \tag{18}$$

A state transition following the new vector field f preserves the first integrals $\boldsymbol{V}$. Because of the above derivation, the cFINDE can be considered as a continuous-time version of a projection method.

The dFINDE is defined as a discretization of the cFINDE, and in that sense, it is a projection method. At the same time, the dFINDE can be considered as a generalization of discrete gradient methods [6, 11, 15, 23, 25, 44].

B.2 Discrete Gradient

A discrete gradient is a discrete analogue to a gradient [6, 11, 15, 23, 25, 44]. Discrete gradients that satisfy Definition 2 are not unique, and many variations have been proposed. For a neural network, Matsubara et al. [38] 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

$$\lim_{||\boldsymbol{h}||_{\mathbb{R}^N} \to 0} \frac{||g(\boldsymbol{u} + \boldsymbol{h}) - g(\boldsymbol{u}) + \mathrm{d}g_{\boldsymbol{u}}(\boldsymbol{h})||_{\mathbb{R}^M}}{||\boldsymbol{h}||_{\mathbb{R}^N}} = 0. \tag{19}$$

The differential $\mathrm{d}g$ acting on a vector $\boldsymbol{w}$ is equivalent to the multiplication of a vector $\boldsymbol{w}$ with the Jacobian $J_g(\boldsymbol{u})$ of the function g at point $\boldsymbol{u}$, that is, $\mathrm{d}g_{\boldsymbol{u}}(\boldsymbol{w}) = J_{g(\boldsymbol{u})}\boldsymbol{w}$. 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. In this way, 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 play the role of the discrete Jacobian. A discrete gradient obtained by the above steps satisfies Definition 2.

B.3 Training Procedures

For the cFINDE, we used the *1-step* error as the loss function. A state $\boldsymbol{u}^{s}_{\mathrm{GT}}$ is taken from the training dataset, and a numerical integrator solves the cFINDE $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{u} = f(\boldsymbol{u})$ for f defined in Eq. (6) and predicts the next state $\boldsymbol{u}^{s+1}_{\mathrm{pred}}$. Then, the cFINDE can be trained to minimize the difference between the predicted state $\boldsymbol{u}^{s+1}_{\mathrm{pred}}$ and the ground truth $\boldsymbol{u}^{s+1}_{\mathrm{GT}}$ taken from the training dataset. Instead of using the state directly, we used the finite difference normalized by the time step size Δt^s for the loss function;

$$\left\| \frac{\boldsymbol{u}^{s+1}_{\mathrm{GT}} - \boldsymbol{u}^{s}_{\mathrm{GT}}}{\Delta t^s} - \frac{\boldsymbol{u}^{s+1}_{\mathrm{pred}} - \boldsymbol{u}^{s}_{\mathrm{GT}}}{\Delta t^s} \right\|_2^2.$$

For the dFINDE, we have defined the state update in Eq. (11). Given a current state $\boldsymbol{u}^s$, the process to obtain the next state $\boldsymbol{u}^{s+1}$ is implicit. Therefore, the prediction by the dFINDE is implicit. However, during the training phase, the ground truth $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ of the next state is known. Hence, we assigned the data points $\boldsymbol{u}_{\mathrm{GT}}^{s}$ and $\boldsymbol{u}_{\mathrm{GT}}^{s+1}$ in the training dataset to both the current state $\boldsymbol{u}^s$ and the next state $\boldsymbol{u}^{s+1}$. Then, the loss function can be the difference between the left- and right-hand sides, that is,

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

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

Some previous studies have proposed alternative loss functions for learning long-term dynamics [10][60, 63]. 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 algorithm, and the dFINDE can adopt it after a minor modification. While it is useful to pursuit the absolute performance, it requires additional hyperparameters such as the length of prediction time and additional effort to adjust them. For simplicity and fair comparisons, we used the 1-step error in the present study.

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 of 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 the 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 the masses of two bodies, respectively. The same process applies to for the remaining positions.

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

$$H = \frac{1}{2}(m_1(v_{x1}^2 + v_{y1}^2) + m_2(v_{x2}^2 + v_{y2}^2)) - \frac{Gm_1m_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}. \tag{20}$$

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

$$\begin{aligned} p_x &= \frac{m_1 v_{x1} + m_2 v_{x2}}{m_1 + m_2}, \\ p_y &= \frac{m_1 v_{y1} + m_2 v_{y2}}{m_1 + m_2}, \end{aligned} \tag{21}$$

and angular momentum [27].

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. Then, the two masses trace elliptical orbits, and trace the exact circular orbits if $\epsilon_v = \epsilon_\theta = 0$. 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 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 [22]. The dynamics is given by

$$u_t = -\alpha u_x + \beta u_{xxx}, \tag{22}$$

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

$$H(u) = \int -\frac{1}{6}\alpha u^3 - \frac{1}{2}\beta u_x^2 \, \mathrm{d}x. \tag{23}$$

As Hamilton's equation $\frac{\mathrm{d}}{\mathrm{d}t}u = S\nabla H$, the partial differential operator $\frac{\partial}{\partial x}$ works 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 $T_2 = \int u^2 \mathrm{d}x$ [39]. Other first integrals are defined using higher-order partial derivatives.

Following the experiments in a previous study [38], we discretized the KdV equation; it no longer has infinitely many first integrals. We set $\alpha = -6$, $\beta = 1$, the spatial size to 10 space units, and the space mesh size to 0.2; the system state $\boldsymbol{u}$ had 50 elements. We generated two solitons as the initial condition; each soliton 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 was set to be at least 2.0 away from each other. We employed the discrete gradient method in [22] to ensure energy conservation.

We set the 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. We trained each model for 30,000 iterations.

Poisson System: Double Pendulum A double pendulum is depicted in in Fig. A1. In polar coordinates, it 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 $\frac{\mathrm{d}}{\mathrm{d}t}\theta_2 = \omega_2$. Given the lengths l_1, l_2 of the two rods, the masses m_1, m_2 of the two weights, and the gravitational acceleration g, the acceleration is given by

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\omega_1 &= \frac{m_2 g \sin\theta_2 \cos\Delta - (l_1\omega_1^2 \cos\Delta + l_2\omega_2^2)m_2 \sin\Delta - (m_1+m_2)g\sin\theta_1}{l_1(m_1 + m_2\sin^2\Delta)}, \\
\frac{\mathrm{d}}{\mathrm{d}t}\omega_2 &= \frac{(m_1+m_2)(l_1\omega_1^2\sin\Delta - g\sin\theta_2 + g\sin\theta_1\cos\Delta) + m_2 l_2 \omega_2^2 \sin\Delta\cos\Delta}{l_2(m_1 + m_2\sin^2\Delta)},
\end{aligned} \tag{24}$$

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 as $x_1 = l_1 \sin\theta_1$, $y_1 = l_1\cos\theta_1$, $x_2 = x_1 + l_2\sin\theta_2$, and $y_2 = y_1 + l_2\cos\theta_2$, and the velocity is transformed accordingly. The total energy H is given by

$$H = \frac{1}{2}(m_1(v_{x_1}^2 + v_{y_1}^2) + m_2(v_{x_2}^2 + v_{y_2}^2)) + g(m_1 y_1 + m_2 y_2). \tag{25}$$

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 are 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.

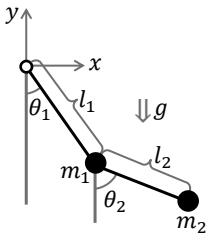

Figure A1: Diagram of the double pendulum.

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 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 Dataset 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 [30] 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 is composed of a resistor R, inductor L, capacitor C, and tunnel diode D 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) gives $I_C + I_D + I_R = I$ and $I_R = I_L$, and Kirchhoff's voltage law (KVL) gives $V_C = V_D = V_R + V_L + E$. 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 $D(V) = V^3/3 - V$. Then, we obtain the FitzHugh–Nagumo model of the original parameters as

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}V &= V - V^3/3 - W + I,\\ \frac{\mathrm{d}}{\mathrm{d}t}W &= 0.08(V + 0.7 - 0.8W).\end{aligned} \tag{26}$$

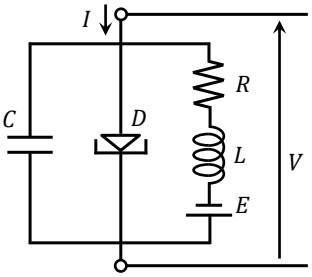

Figure A2: Circuit diagram of the FitzHugh–Nagumo model [30].

Due to the resister 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 is 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 [53]. 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 [20, 57].

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 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 Symbolic Regression of Learned First Integrals

Using gplearn, we performed a symbolic regression of the first integrals $\boldsymbol{V}$ learned by the neural network. Gplearn is based on genetic programming. We prepared addition $+$, subtraction $-$, multiplication $\times$, 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

Table A1: Symbolic Regression of First Integrals Learned from 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. $\alpha = 0.003(y_1+y_2)(v_{x2}+x_1+y_1(v_{x2}+y_1+y_2)+1.402)$.

set the other hyperparameters to their default values, e.g., the maximum number of generations was 20.

We summarized regression results of the HNN with cFINDE for $K = 2$ trained using the 2-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. (21). 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, or they can be regarded as the linear momenta in the $(1, -1)$-direction and in the $(1, 1)$-direction, respectively.

When the quantities $V_1(\boldsymbol{u})$ and $V_2(\boldsymbol{u})$ are first integrals, any function only of $V_1(\boldsymbol{u})$, $V_2(\boldsymbol{u})$, and arbitrary constants is a first integral functionally dependent on $V_1(\boldsymbol{u})$ and $V_2(\boldsymbol{u})$. Thus, in general, there is no guarantee that FINDE will find first integrals in their well-known forms. However, recent studies have revealed that typical initialization and training procedures of neural networks tend to learn simple functions [3, 5]. Additionally, the symbolic regression limited the depth of the computation graph, biasing the results toward simple functions. This is why the learned first integrals were often 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 of the range of training data. Once first integrals are learned 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.

D.2 With Known First Integrals

The double pendulum is classified as a constrained Hamiltonian system. CHNN was proposed for cases when holonomic constraints are known [20]. 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 first integrals, but did not work properly. The original purpose of projection methods is to eliminate numerical errors of first integrals, but not to change the class to which the dynamics belong. Therefore,

Table A2: Results with known holonomic constraints.

	2-pend		**2-body**	
Model	**1-step**↓	**VPT**↑	**1-step**↓	**VPT**↑
NODE	0.82 ±0.02	0.110 ±0.035	144.21 ±12.65	0.134 ±0.014
HNN [26]	6220.26 ±91.57	0.002 ±0.000	5.17 ±0.57	0.362 ±0.026
CHNN [20]	0.07 ±0.00	0.928 ±0.036	(not working)	
NODE+cFINDE	0.71 ±0.04	0.461 ±0.071	163.64 ±9.79	0.147 ±0.024
HNN+cFINDE	236.51 ±7.15	0.020 ±0.002	8.32 ±0.43	0.476 ±0.040

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.

On the other hand, 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 get the inverse of a singular matrix and could not even learn the dynamics. In contrast, the cFINDE improved the performances of both NODE and HNN.

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.

D.3 First Integral Preservation for Hamiltonian System

In Fig. 1, we examined an ODE of mass-spring system and FINDE using the leapfrog integrator. Here, we also examined the case with the Dormand–Prince integrator 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 in the bottom panel. Even using the Dormand–Prince integrator, which is a fourth-order method, the energy is slightly decreasing. The cFINDE with the Dormand–Prince integrator shows the same tendency. This phenomenon is due to numerical errors and is called energy drift. The dFINDE with the Dormand–Prince integrator significantly suppresses the energy error. The remaining error is caused by rounding errors.

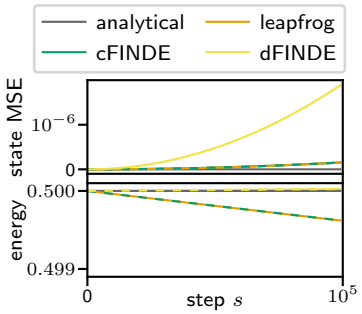

Figure A3: Integration of a known mass-spring system.

When the focus is on the MSEs of the state in the upper panel, the trend is different; The dFINDE with the Dormand–Prince integrator suffers from larger state errors. Although the dFINDE is designed to eliminate errors of the energy, it does not necessarily minimize state errors. The Dormand–Prince integrator, on the other hand, is designed to suppress state errors.

Therefore, there is no guarantee that the dFINDE improves the prediction performance, which is defined using state errors. However, the experimental results in Table 3 demonstrate that the dFINDE is superior to the base model and the cFINDE in VPT. For the mass-spring system, the governing equation is already known as an ODE, and it is discretized by the dFINDE, leading to discretization errors. On the other hand, 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, neither the ODE nor the discretization is involved, no discretization error occurs, and we only the advantage of exactly preserving the first integral.

This kind of paradox has been repeatedly discovered in previous studies. For example, the leapfrog integrator and the discrete gradient method are second-order methods, but they are superior to the Dormand-Prince integrator when being combined with neural networks and learning dynamics from data [38]. For learning, the preservation of specific properties of target systems is more important than the order of accuracy.

References

[58] Alet, F., Doblar, D., Zhou, A., Tenenbaum, J., Kawaguchi, K., and Finn, C. (2021). Noether Networks: Meta-Learning Useful Conserved Quantities. *Advances in Neural Information Processing Systems (NeurIPS)*, (NeurIPS):1–20.

[59] Celledoni, E., Leone, A., Murari, D., and Owren, B. (2022). Learning Hamiltonians of Constrained Mechanical Systems. *arXiv*, pages 1–18.

[60] Course, K. L., Evans, T. W., and Nair, P. B. (2020). Weak form generalized Hamiltonian learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, number NeurIPS.

[61] Shen, Y., Gu, J., Tang, X., and Zhou, B. (2020). Interpreting the latent space of GANs for semantic face editing. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9240–9249.

[62] Yang, S., He, X., and Zhu, B. (2020). Learning Physical Constraints with Neural Projections. *Advances in Neural Information Processing Systems*, pages 1–15.

[63] Zhong, Y. D., Dey, B., and Chakraborty, A. (2020). Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *International Conference on Learning Representations (ICLR)*, pages 1–17.