Hamiltonian Neural Networks for Solving Differential Equations

Marios Mattheakis, David Sondak, Akshunna S. Dogra, and Pavlos Protopapas

Abstract—There has been a wave of interest in applying machine learning to study dynamical systems. In particular, neural networks have been applied to solve the equations of motion, and therefore, track the evolution of a system. In contrast to other applications of neural networks and machine learning, dynamical systems -depending on their underlying symmetries- possess invariants such as energy, momentum, and angular momentum. Traditional numerical iteration methods usually violate these conservation laws, propagating errors in time, and reducing the predictability of the method. We present a Hamiltonian neural network that solves the differential equations that govern dynamical systems. This unsupervised model is learning solutions that satisfy identically, up to an arbitrarily small error, Hamilton's equations and, therefore, conserve the Hamiltonian invariants. Once it is optimized, the proposed architecture is considered a symplectic unit due to the introduction of an efficient parametric form of solutions. In addition, by sharing the network parameters and the choice of an appropriate activation function drastically improve the predictability of the network. An error analysis is derived and states that the numerical errors depend on the overall network performance. The symplectic architecture is then employed to solve the equations for the nonlinear oscillator and the chaotic Hénon-Heiles dynamical system. In both systems, the symplectic Euler integrator requires two orders more evaluation points than the Hamiltonian network in order to achieve the same order of the numerical error in the predicted phase space trajectories.

Index Terms—Hamiltonian neural network, unsupervised model, symplectic architecture, nonlinear dynamical systems, chaotic motion

I. INTRODUCTION

STUDYING the evolution of dynamical systems has become a significant trend in scientific research. The information age has generated an exponential increase in the amount of digital data being stored, and a non-trivial fraction of these data-sets describe the evolution of dynamical systems. These include a wide range of systems, from large-scale astrophysics to nano-scale quantum physics. Recently, machine learning models, and particularly neural networks (NNs), have been used to explore those data and forecast the future behavior of complex dynamical systems [1]–[5], improve turbulence

M. Mattheakis is with John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA (e-mail: mariosmat@.seas.harvard.edu) (see https://scholar.harvard.edu/marios_matthaiakis).

A. S. Dogra is with the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA, and with the Department of Physics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA.

D. Sondak, and P. Protopapas are with John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA.

models [6]–[9], discover differential equations (DEs) [10]–[13], and find approximate solutions for those equations [14], [15]. In addition to the data-driven studies, equation-driven unsupervised NNs have been used to solve ordinary and partial DEs relevant to a variaty of physical systems [16]–[19]. Equation-driven networks construct analytical functions that satisfy a particular differential structure; subsequently, in the training process of such models, we do not need any ground truth data. Essentially, the loss function solely depends on the solutions obtained by the NN. Furthermore, the universal approximation theorem of NNs [20] states that a NN can approximate any function with arbitrary accuracy. This makes NNs as a suitable approach to solving complicated problems governed by differential equations.

This work presents a Hamiltonian neural network architecture that is used to solving DE systems. The Hamiltonian NN is an evolution of previously used unsupervised NNs for finding solutions to DEs that satisfy boundary and initial conditions. We improve the NN DE solvers by speeding the convergence of the network to the solution while reaping the benefits of the underlying physical properties. We propose a NN architecture inspired by and geared towards Hamiltonian systems with time-independent Hamiltonians. Once optimized, the NN satisfies Hamilton's equations over the entire temporal domain, directly implying the conservation of every invariant under the respective Hamiltonian flow. The proposed Hamiltonian NNs consist of a more numerically precise and robust method to solve dynamical equations than standard semi-implicit schemes such as a symplectic Euler integrator. By sharing the network weights, choosing a trigonometric activation function, and using an efficient parametric form of solutions, we show a speedup in the convergence behavior during the optimizing process and, subsequently, an improvement in the predictability of the network. Also, it shown that the proposed NN architecture can be considered a true and globally symplectic and thus, time invariant unit.

In the rest of this study, we describe the Hamiltonian NN architecture that is used to approximate Hamiltonian trajectories. An error analysis is performed and shows that the accuracy of the predicted solutions can be predefined before optimizing the network. Then, the proposed symplectic NN is applied to solve the equations that describe the motion of a nonlinear oscillator and a two-dimensional chaotic system. We point out situations where the Hamiltonian NN solver out-performs the symplectic Euler integrator. The network performance is demonstrated by exploring different architectures through different parametric solutions and activation functions. We conclude this study with a summary of the key ideas introduced in this work,

the advantages of using Hamiltonian NN to solving DEs, and with a discussion of future plans.

II. METHOD

A cornerstone idea in classical mechanics is that a system's evolution can be investigated through the study of its underlying symmetries and constraints. By the 20th century, Lagrange, Hamilton, and others had shown that the dynamics of a system is tethered to simple scalar functions, the Lagrangian and Hamiltonian functions, with multiple conservation laws and their underlying symmetries prepackaged with these functions. These scalar functions are then used to derive the DEs that govern the spatiotemporal motion of a system. In particular, starting from the Lagrangian (the difference between kinetic and potential energy), invoking Hamilton's principle (the motion follows trajectories that minimize the action integral), and employing techniques from the calculus of variations, the motion of a system is described by Euler-Lagrange (E-L) equation. In the Hamiltonian formulation, on the other hand, we start from the Hamiltonian which is a transformation of the Lagrangian and is a conservative quantity, namely it does not change in time. This formulation results in Hamilton's equations, which are equivalent to the E-L equation and therefore minimize the same action. The Hamilton's equations are a coupled set of first order DEs, whereas, Lagrangian formalism provides a single set of second-order DEs. The Hamiltonian formulation possesses inherent advantages over the Lagrangian as a coupled set of first-order DEs is numerically more stable and more comfortable to solve than a single set of second-order DEs. The resulting DEs are often analytically intractable, so engineers and scientists resort to discretization techniques to obtain solutions. However, the discretization procedure for solving the DEs could lead to violations of the underlying conservation laws. This issue can by cured by using NN solvers that able to provide analytical solutions that respect the underlying principles. Indeed, any sort of semi-implicit method, like symplectic Euler integrator, allows errors to build and blow up in time. Chaotic systems in particular, are highly sensitive to such concerns and are, therefore, ideal ground for testing the performance of the proposed Hamiltonian NN.

We consider a physical system of many bodies that are moving in space. The spatio-temporal motion of those objects can be described in a d-dimensional configuration space which is defined by the specification of the position as a function of the time t of all objects in a system. More precisely, d is defined as the product of the number of bodies in a system and the number of spatial dimensions that those objects are allowed to move. In the Lagrangian formulation we are working on the configuration space, whereas, the Hamiltonian formalism is defined in the phase space, which consists of the position and momentum of the objects. Subsequently, each dimension in the configuration space associates with two degrees of freedom in the phase space. In this work, we are interested in Hamiltonian framework, therefore we consider a phase space of $D = 2d$ dimensions. Many classical systems, from the simple pendulum to solar systems, can be described by the separable Hamiltonian form $\mathcal{H} = T + V$, where the

potential energy term V depends solely on the *generalized space coordinates* $\mathbf{q} = (q_1, \ldots, q_d)$, and the kinetic term T depends solely on the *generalized momenta* $\mathbf{p} = (p_1, \ldots, p_d)$. Since this Hamiltonian form does not depend directly on time, systems described by it will conserve energy. Other dynamical invariants may also be inbuilt, depending upon the specific choice of the individual phase space variables and their corresponding continuous symmetries [21]. As an example, when the Hamiltonian does not directly depend on a coordinate q_i, the associated momentum p_i is conserved and vice versa. For such Hamiltonian functions, the dynamics are governed by following coupled DEs, called Hamilton's or *canonical* equations:

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i}, \qquad \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i}, \qquad (1)$$

where dots denote time derivatives. An elegant way of expressing Hamilton's equations is the *symplectic* notation. Let $\mathbf{z} = (q_1, \ldots, q_d, p_1, \ldots, p_d)^T \in \mathbb{R}^D$, and $\mathbf{J}$ be the $D \times D$ matrix

$$\mathbf{J} = \begin{pmatrix} \mathbf{0} & \mathbf{1} \\ -\mathbf{1} & \mathbf{0} \end{pmatrix}, \qquad (2)$$

where $\mathbf{0}$ and $\mathbf{1}$ represent the $d \times d$ zero and unity matrix, respectively. Then, Hamilton's equations can be written in the compact vector form

$$\dot{\mathbf{z}} = \mathbf{J} \cdot \nabla_{\mathbf{z}} \mathcal{H}(\mathbf{z}), \qquad (3)$$

where $\nabla_{\mathbf{z}} \mathcal{H}(\mathbf{z}) = \partial \mathcal{H}(\mathbf{z}) / \partial \mathbf{z}$. Numerical methods that evaluate Eq. (3) are called symplectic methods and have been widely used to calculate the long-term evolution of chaotic systems [22]. In this work we present an alternative method based on NNs to solve Eq. (3). As we will discuss below, symplectic integrators conserve a Hamiltonian which is slightly perturbed from the original, whereas, symplectic NNs conserve the original Hamiltonian. This is a great advantage that the proposed NN has over the symplectic integrators.

An alternative approach to the numerically solving DEs is offered by feed-forward NNs [16], [17]. One key advantage of such NNs over traditional numerical methods is that they seek to learn actual functions that satisfy the DEs, rather than creating an approximation to the real solution. Moreover, the NN's solutions are in a closed, differentiable, and analytic form [16], and the calculations can be efficiently implemented on parallel architectures leading to significant speed-ups [16]. The advantage in using our proposed NN architecture is that it provides solutions that satisfies Hamilton's equation simultaneously. Thus, the dynamical invariants of a particular Hamiltonian are being identically respected to the required precision, compared to the accumulation of errors that is inevitable in iterative solvers. To compare, we present the semi-implicit Euler method, which is the simplest, yet most widely used, symplectic integrator for solving Hamilton's equation. Symplectic Euler method conserves energy up to a fluctuating error because it conserves a slightly different Hamiltonian than the original. For the separable Hamiltonian

form $\mathcal{H} = T(p_i) + V(q_i)$, the symplectic Euler scheme for solving the system (1) reads

$$q_i^{(n+1)} = q_i^{(n)} + \Delta t \frac{\partial T\left(p_i^{(n)}\right)}{\partial p_i^{(n)}}, \qquad (4)$$

$$p_i^{(n+1)} = p_i^{(n)} - \Delta t \frac{\partial V\left(q_i^{(n+1)}\right)}{\partial q_i^{(n+1)}}. \qquad (5)$$

Here, Δt is the time step between two sequential time points, (n) denotes the time point that is evaluated, $q_i^{(n)} = q_i(n\Delta t)$, and $p_i^{(n)} = p_i(n\Delta t)$. Due to the iterating nature of symplectic Euler method, we read in Eqs. (4), (5) that the solutions at two sequential time points are needed to evaluate Hamilton's equations at any point, leading to numerical error in the calculation of energy, that is proportional to Δt.

The objective of this study is to solve Hamilton's equations (3) by using NNs. Let us consider the general form of parametric solutions

$$\hat{\mathbf{z}}(t) = \mathbf{z}(0) + f(t)\mathbf{N}(t), \qquad (6)$$

where $\hat{\mathbf{z}}$ is the solution vector discovered by the NN, $\mathbf{z}(0)$ is the initial state vector, and $\mathbf{N}(t) \in \mathbb{R}^D$ is a vector of D outputs of a feed-forward fully connected NN. The parametric function $f(t)$ enforces the initial conditions in the parametric solutions, i.e. $\hat{\mathbf{z}}(0) = \mathbf{z}(0)$ when $f(0) = 0$. The network takes as a single input the time point t_n, where n denotes the n-th sequential point; without losing the generality, we consider the initial time $t_0 = 0$. We train the NN by minimizing, with respect to the learning parameters of the network, the mean-squared error (MSE) that is defined for Hamilton's equations (3) as:

$$L = \frac{1}{K} \sum_{n=1}^{K} \left(\dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(n)}} \mathcal{H}\left(\hat{\mathbf{z}}^{(n)}\right) \right)^2, \qquad (7)$$

where $\hat{\mathbf{z}}^{(n)} = \hat{\mathbf{z}}(t_n)$ and K is the total number of the n-points used for the network optimization.

The time derivatives are obtained by using automatic differentiation that computationally costs one back-propagation through the entire network [23]. We first generate equally-spaced time points t_n in the time interval $[0, T]$. Then, we randomly perturb these points in each epoch as: $t_n \rightarrow t_n + \epsilon$ where ϵ is a random term obtained by a normal distribution. This trick improves the network predictability as it is effectively trained over a continuous time interval. In addition, perturbing the training points in every epoch employs the stochastic gradient descent (SGD) method, and thus it assists the optimizer to escape from local minima in the loss function. Perturbing the points in every epoch means that we perturb the loss function and, subsequently, the local minima are dynamically moving. In the context of SGD, each epoch is considered as a mini-batch while all the epochs consist the whole batch for the training set. Minimizing the loss function in Eq. (7) yields solutions that identically respect the symplectic structure of Eq. (3) and accordingly, every dynamical invariant of the Hamiltonian flow is respected too. The proposed Hamiltonian NN architecture is graphically demonstrated in Fig. 1.
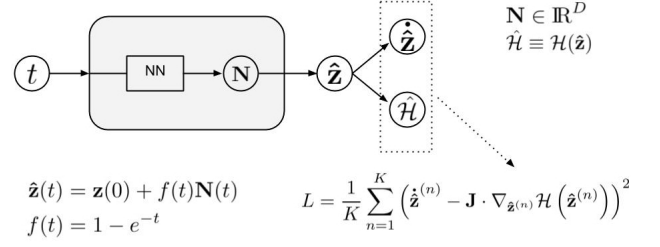

$$\hat{\mathbf{z}}(t) = \mathbf{z}(0) + f(t)\mathbf{N}(t)$$
$$f(t) = 1 - e^{-t}$$

$$\mathbf{N} \in \mathbb{R}^D$$
$$\hat{\mathcal{H}} \equiv \mathcal{H}(\hat{\mathbf{z}})$$

$$L = \frac{1}{K} \sum_{n=1}^{K} \left(\dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(n)}} \mathcal{H}\left(\hat{\mathbf{z}}^{(n)}\right) \right)^2$$

Fig. 1. Hamiltonian architecture with parametrization $\hat{\mathbf{z}}(t)$ used in the loss function L; $\mathcal{H}$ is the Hamiltonian and $f(t)$ imposes the initial conditions to $\hat{\mathbf{z}}(t)$; K is the number of the training points and (n) indicates each time point.

A crucial role in the performance of the network is played by $f(t)$. In [16], authors use $f(t) = t$, which satisfies $f(0) = 0$. However, this is an unbounded function that adds further difficulty when t becomes large. Specifically, for the NN outputs after enough epochs, Eq. (6) states that $\mathbf{N} = (\hat{\mathbf{z}} - \mathbf{z}(0))/t$. As t increases the $\mathbf{N}$ tends to zero, which affects negatively the network predictability in large time scales. To remedy this inefficiency we propose the parametric function

$$f(t) = 1 - e^{-t}, \qquad (8)$$

which is a smooth, bounded function, with $f(0) = 0$. Later, we show that the specific choice of parametric function drastically improves the predictability of the NN solver. Interestingly enough, the fact that $f(t)$ rapidly tends to 1 implies that the proposed architecture consists a symplectic NN. In particular, at the limit $L \rightarrow 0$ Eq. (7) yields $\dot{\hat{\mathbf{z}}} = \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}} \mathcal{H}(\hat{\mathbf{z}})$, and as $t \rightarrow \infty$, we have $\hat{\mathbf{z}} = \mathbf{z}(0) + \mathbf{N}$. Considering the afore-mentioned two limits and performing the linear transformation $\mathbf{N} \rightarrow \mathbf{N} - \mathbf{z}(0)$ we obtain:

$$\dot{\mathbf{N}} = \mathbf{J} \cdot \nabla_{\mathbf{N}} \mathcal{H}(\mathbf{N}), \qquad (9)$$

which indicates that the proposed architecture comprises a symplectic NN that states that the function $\mathcal{H}(\mathbf{N})$ is time invariant.

An important distinction between the present and previous method presented in [16] is that the latter proposes one NN per DE, and therefore, D single-output networks are required. This is conceptually different from our approach, where we suggest one NN with D outputs. By using a single network, the individual outputs share all the weights except those in the output layer, allowing correlations between the outputs. Hamilton's equations are indeed correlated and thus, by sharing the weights assists the network to discover these co-dependencies. Consequently, we obtain the same complexity with fewer learning parameters, yielding a more robust and efficient network. We point out a set of substantial differences between our study and Lagaris' et al. work [16]: the depth of the NN and the choice of the activation function. In [16], the network architecture is restricted to one hidden layer, uses sigmoid activation functions, and the derivatives used for the back-propagation are calculated analytically. In the current work, we use automatic differentiation to calculate the derivatives [23]. Subsequently, we were able to use any

activation function and arbitrarily many hidden layers, which leads to greater control over the complexity of the network. We choose the trigonometric $\sin(\cdot)$ as the activation function and show that the NN converges to the solutions with less training iterations than using the sigmoid activation. Using $\sin(\cdot)$ as the activation function permits the solution to be expressed on a basis that has global support similar to the Fourier series. The derivative of $\sin(\cdot)$ has more global support than the derivatives of the traditional activation functions, such as those from the sigmoid family, hence $\sin(\cdot)$ are a more expressive activation function. Moreover, since $\sin(\cdot)$ is a periodic function, it introduces multiple periodic local minima in the loss function. Empirically, we find that these local minima perform equally well and as a result, it is easier to reach a minimum during the optimization process.

An alternative deep learning approach to solve the mechanical equations is given in the context of the Lagrangian formulation. In that case we have to solve a system of d second order ordinary DEs and a parametrization that enforces the initial conditions will be:

$$\hat{\mathbf{q}}(t) = \mathbf{q}(0) + f_1(t)\dot{\mathbf{q}}(0) + f_2(t)\mathbf{N_L}(t), \quad (10)$$

with the constraints $f_1(0) = 0$ and $f_2(0) = \dot{f}_2(0) = 0$, and $\mathbf{N_L}$ is vector that consists of the d outputs of a feed-forward NN with $\mathbf{N_L}(t) \in \mathbb{R}^d$. The $\hat{\mathbf{q}} = (\hat{q}_1, \ldots, \hat{q}_d)^T$ is the predicted solutions for the position while $\mathbf{q}(0)$ and $\dot{\mathbf{q}}(0)$ are the initial position and velocity vector states, respectively. The loss function to be minimized is once again the MSE and is defined by an E-L equation by assuming a Lagrangian of the form $\mathcal{L} = T - V$ with $T = \dot{\mathbf{q}}^2/2$ and $V = V(\mathbf{q})$. Thence, the loss function reads:

$$L = \frac{1}{K} \sum_{n=1}^{K} \left(\ddot{\hat{\mathbf{q}}}^{(n)} + \nabla_{\hat{\mathbf{q}}}^{(n)} V\left(\hat{\mathbf{q}}^{(n)}\right) \right)^2, \quad (11)$$

where $\hat{\mathbf{q}}^{(n)} = \hat{\mathbf{q}}(t_n)$. The choice of the parametric functions $f_1 = t$ and $f_2 = t^2$ is the situation discussed in [16], however, we find that $f_1 = 1 - e^{-t}$ and $f_2 = (1 - e^{-t})^2$ yield a better network performance for the same reasons mentioned above for the Hamiltonian NN. Although the Lagrangian NN converges to the same solutions with the Hamiltonian NN, it is a less computational efficient. This is because the Hamiltonian network of the present work solves first order DEs and, therefore, only one back-propagation is required to calculate each derivative. Networks that solve higher order equations require additional back-propagations, which drastically increases the computational footprint in the context of memory and floating point operations. Indeed, in our calculations we observe that for the same DE system, the optimization for a Lagrangian NN requires about twice as much computational time as a Hamiltonian network.

III. ERROR ANALYSIS

We seek to provide a rough bound on the error in the solution based on the maximum value of the loss function. To begin, note that Eq. (7) can be written as $L = \sum_n \ell_n^2/K$ where

$$\ell_n = \dot{\hat{\mathbf{z}}}^{(n)} - \mathbf{J} \cdot \nabla_{\hat{\mathbf{z}}^{(n)}} \mathcal{H}\left(\hat{\mathbf{z}}^{(n)}\right) \quad (12)$$

is a vector $\ell_n = (\ell_{n,1}, \ldots, \ell_{n,D})$ that contains all the loss components at some arbitrary time point t_n. Since L is the loss function for the NN, averaged over time, ℓ_n^2 can be considered the instantaneous loss at the nth time point. Let $\delta\mathbf{z} = \mathbf{z} - \hat{\mathbf{z}}$ be the error between the true solution and the NN solution. Expanding the Hamiltonian $\mathcal{H}(\mathbf{z}) = \mathcal{H}(\hat{\mathbf{z}} + \delta\mathbf{z})$ in a Taylor series about $\hat{\mathbf{z}}$ and keeping up to quadratic terms yields:

$$\mathcal{H}(\mathbf{z}) \approx \mathcal{H}(\hat{\mathbf{z}}) + (\nabla_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z} + \frac{1}{2}(\boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}^2, \quad (13)$$

where $\boldsymbol{\mathcal{D}}_{\mathbf{z}}$ is the Hessian matrix. Taking the gradient of Eq. (13) with respect to $\mathbf{z}$ and rearranging terms gives,

$$(\nabla_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \approx \nabla_{\mathbf{z}}\mathcal{H}(\mathbf{z}) - (\boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\hat{\mathbf{z}}))_{\mathbf{z}} \delta\mathbf{z}. \quad (14)$$

We note that for Hamiltonians with quadratic dependence on $\mathbf{z}$, the quadratic expansion (13) is exact because higher order terms vanish. In addition, the second order in $\delta\mathbf{z}$ is the smallest order still large enough to not be canceled when we move to substitute Eq. (14) into Eq. (12). Nevertheless, the derivation can be extended to include higher order terms in a straightforward manner. In what follows, we drop the superscript (n) for clarity of presentation. Substituting the Taylor series expansion (14) into (12) and invoking (3) results in,

$$\ell \approx \mathbf{J} \cdot [(\boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}] - \dot{\delta\mathbf{z}}. \quad (15)$$

Inspecting the vector DE (15) we read that its components comprise a closed differential system for the error δz_i in each predicted trajectory $\hat{z}_i$. Solving this differential system with initial condition $\delta\mathbf{z}(0) = 0$, as it is dictated by the parameterization (6), we can compute how the errors propagate in time. However, this requires knowledge of the loss components of $\ell(t)$ and thus, such an analysis can be performed only after we have trained the network.

On the other hand, we can derive a bound on the size of $\delta\mathbf{z}$ without having exact knowledge of $\ell(t)$ by constructing a worst case scenario. We want to establish a relationship between ℓ and $\delta\mathbf{z}$, such that it determines when to stop the network training in order to get solutions with better than a certain accuracy. Let $\ell_{\max}^2 = \max_t(\ell^2)$ represents the largest instantaneous loss that the neural network will have after being trained. In the following analysis, we denote the $2-$ norm by $\|\cdot\|$. We have,

$$
\begin{aligned}
\ell_{\max}^2 &\geq \|\dot{\delta\mathbf{z}} - \mathbf{J} \cdot (\boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}\|^2 \\
&\geq \left| \|\dot{\delta\mathbf{z}}\| - \|(\mathbf{J} \cdot \boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}\| \right|^2 \\
&= \|\dot{\delta\mathbf{z}}\|^2 - 2\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J} \cdot \boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}\| + \|(\mathbf{J} \cdot \boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}\|^2 \\
&\geq \|\dot{\delta\mathbf{z}}\|^2 - 2\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J} \cdot \boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}} \delta\mathbf{z}\| + (\sigma_{\min}\|\delta\mathbf{z}\|)^2,
\end{aligned}
\quad (16)
$$

where $\sigma_{\min}$ is the minimum singular value of $(\boldsymbol{\mathcal{D}}_{\mathbf{z}}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}$. The last line in the above expression (16) can be obtained by considering the quantity $\|Ax\|$ and using the singular

value decomposition on A to show that $\|Ax\| \geq \sigma_{\min}\|x\|$. Rearranging terms leads to,

$$\sigma_{\min}^2\|\delta\mathbf{z}\|^2 \leq \ell_{\max}^2 - \|\dot{\delta\mathbf{z}}\|^2 + 2\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J}\cdot\boldsymbol{\mathcal{D}_z}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}\,\delta\mathbf{z}\|$$
$$\leq \ell_{\max}^2 - \|\dot{\delta\mathbf{z}}\|^2 + 2\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J}\cdot\boldsymbol{\mathcal{D}_z}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}\|\|\delta\mathbf{z}\|$$
$$\Rightarrow \sigma_{\min}^2\|\delta\mathbf{z}\|^2 - 2\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J}\cdot\boldsymbol{\mathcal{D}_z}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}\|\|\delta\mathbf{z}\| \leq \ell_{\max}^2 - \|\dot{\delta\mathbf{z}}\|^2.$$
(17)

Solving the above quadratic inequality (17) for $\|\delta\mathbf{z}\|$ yields,

$$\|\delta\mathbf{z}\| \leq \frac{\|\dot{\delta\mathbf{z}}\|\|(\mathbf{J}\cdot\boldsymbol{\mathcal{D}_z}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}\|}{\sigma_{\min}^2}$$
$$+\frac{1}{\sigma_{\min}^2}\left[\sigma_{\min}^2\ell_{\max}^2 - \|\dot{\delta\mathbf{z}}\|^2\left(\sigma_{\min}^2 - \|(\mathbf{J}\cdot\boldsymbol{\mathcal{D}_z}\mathcal{H}(\mathbf{z}))_{\hat{\mathbf{z}}}\|^2\right)\right]^{1/2}.$$
(18)

Now consider a single component of the error, δz_i. The largest value δz_i can take occurs when $\delta z_i \neq 0$ and $\delta z_j = 0$ for $j \neq i$. That is, for a fixed error, all of the error is concentrated in a single component. In this case, $\|\delta\mathbf{z}\|^2 = \delta z_i^2$. If δz_i^2 is maximized at a value $t_{\max}$, then $(\dot{\delta z_i^2}) = 0$ at $t_{\max}$. Therefore, $\delta z_i\dot{\delta z_i} = 0 \Rightarrow \dot{\delta z_i} = 0$. Using this in (18) provides,

$$\|\delta z_i\| \leq \frac{\ell_{\max}}{\sigma_{\min}}.$$
(19)

If a NN is trained such that the loss function has a maximum value of $\ell_{\max}$, then the maximum error that any component of the solution can take is bounded by (19). In other words, we can choose in advance an accuracy for the solutions and use the relationship (19) to calculate the $\ell_{\max}$, which, therefore, will determine when we have to stop training the network ensuring the desirable accuracy. The $\sigma_{\min}$ can be calculated due to the training process since, in the most general case, it is a function of the solutions. Moreover, the expressions (15) and (19) state that $|\delta\mathbf{z}|$ depends on the general network performance and not only on the number of the time points used in the training process, which is the case of numerical integrators. That happens because the number of training points is not the only parameter that determines the value of the loss function. For example, fixing the number of the training points while increasing the number of hidden layers or neurons yields better performance that corresponds to a smaller $\ell_{\max}$. In summary, once the Hamiltonian NN is optimized, Eq. (15) can be used to calculate the error propagation. On the other hand, we can decide the accuracy of the solutions before the optimization by using Eq. (19) to define the $\ell_{\max}$ that determines when to stop training the network.

IV. NONLINEAR OSCILLATOR

As a concrete example, we consider the one dimensional nonlinear (an-harmonic) oscillator with Hamiltonian

$$\mathcal{H} = \frac{p^2}{2} + \frac{x^2}{2} + \frac{x^4}{4},$$
(20)

where the natural frequency and the mass of the oscillator are considered to be unity. The Hamiltonian (20) corresponds to the total energy E of the system, and the associated equations of motion read (Eq. 1):

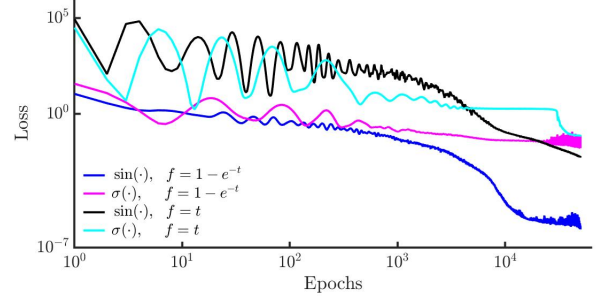$$\dot{x} = p, \qquad \dot{p} = -(x + x^3).$$
(21)

Fig. 2. Hamiltonian NN solves the equations of the nonlinear oscillator system. Color lines represent the loss function in log-scale during the training for different combinations of activation and parametric functions f shown in legend.

In what follows, we use the symplectic NN architecture to solve the above nonlinear Hamiltonian system and compare the NN solutions with those obtained by symplectic Euler integrator. It results that the symplectic Euler method requires two orders more evaluation time points than the NN to reach the same numerical error. We also explore the efficiency of the network for different activation and parametric functions.

The phase space of the oscillator consists of two degrees of freedom with $\mathbf{z} = (x, p)^T$. Accordingly, we utilize a feed-forward NN with two outputs $\mathbf{N} = (N_1, N_2)^T$ used to parametrize the approximate solutions $\hat{\mathbf{z}} = (\hat{x}, \hat{p})^T$ according to Eq. (6). The loss function is defined by Eqs. (21) and according to Eq. (7) as:

$$L = \frac{1}{K}\sum_{n=0}^{K}\left[\left(\dot{\hat{x}}^{(n)} - \hat{p}^{(n)}\right)^2 + \left(\dot{\hat{p}}^{(n)} + \hat{x}^{(n)} + \left(\hat{x}^{(n)}\right)^3\right)^2\right].$$
(22)

We initialize a grid with $K = 200$ time points equally spaced in the time interval $t = [0, 4\pi]$. At the beginning of each epoch, we perturb all the time points by using a random term obtained by a normal distribution with zero mean and a standard deviation of 0.06π. The initial state is chosen to be $(x_0, p_0) = (1.3, 1.0)$, corresponding to the total initial energy $E_0 = 2.06$; in this energy, the motion deviates from the behavior of the simple harmonic oscillator. The NN consists of two hidden layers with 50 neurons per hidden layer, and is being trained for $5\cdot10^4$ epochs by using Adam optimizer [24] with a learning rate of $8\cdot10^{-3}$. We perform four independent numerical experiments that correspond to different NN designs, namely for the combinations of sigmoid $\sigma(\cdot)$ and trigonometric $\sin(\cdot)$ activation functions, and for the parametric functions $f(t) = t$ and $f(t) = 1 - e^{-t}$. Figure 2 demonstrates in logarithmic scale the loss function (22) during the training; each color represents one of the the four distinguished cases of architectures according to the legend. We highlight that the loss function of our proposed design (blue line) converges faster than the other models.

The performance of the Hamiltonian NN after its training is represented in Fig. (3) by the blue curve. In addition, we use the DEs solver `odeint` of the `scipy` python package [25] to solve the system (21) and consider the obtained numerical solutions as the ground truth. We note that the solvers provided
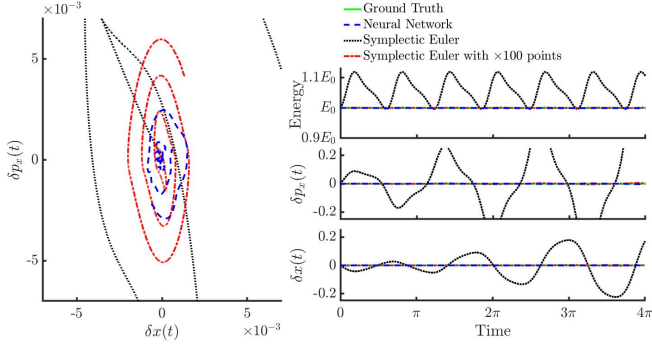
Fig. 3. Comparing the ground truth (green) with the approximated solutions obtained by NN (blue) and by symplectic Euler integrator. The NN is trained over $K = 200$ time points while the integrator is evaluated at K (black) and $100 \times K$ (red) points. Left: The phase space of the numerical error. Right: The error evolution in position and momenta, and the total energy in time.

by `scipy` have exemplary error control and adaptivity leading to excellent solution trajectories. For comparison purposes, we also utilize the symplectic Euler method described in Eqs. (4),(5) to solve the DEs (21), and compare the solutions with those obtained by our proposed symplectic NN. In Fig. 3 we present results obtained by the solver (green lines), by the NN (blue line), and by the symplectic Euler integrator (black and red). After the network optimization we get $\ell_{\max} = 3.3 \cdot 10^{-3}$. The smallest singular value of the Hessian of Hamiltonian (20) is $\sigma_{\min} = 1$. Subsequently, Eq. (19) yields for both δx and δp the upper bound error $3.3 \cdot 10^{-3}$. Interestingly enough, the symplectic Euler method needs $100 \times K$ time points to approach this maximum error. In the case of Euler's method, we present in Fig. 3 two numerical solutions: one with the same time points K used in the NN training (black), and a second with 100 times more points (red). The left graph in Fig. 3 demonstrates the phase space for the numerical errors where we observe that the errors in the NN's solutions are in the same order with the error obtained by the symplectic Euler when 100 times more time points are used. On the right panel of Fig. 3 we present $\delta x(t)$ and $\delta p(t)$ and the the total energy as a function of time calculated by using the numerical solutions in the Hamiltonian (20). An important result of this exploration is that, in contrast to the Euler integrator, the NN's solutions conserve the total energy locally. This is a consequence of the fact that the solutions obtained by the symplectic NN conserve the correct Hamiltonian rather than a perturbed one, which is the case with the symplectic integrators. Therefore, in the context of the energy conservation task, the Hamiltonian NN outperforms the symplectic Euler integrator.

V. Chaotic system

We demonstrate further the efficiency of the proposed symplectic NN by solving the equations for a chaotic two-dimensional dynamical system. In particular, we solve the canonical equations for the Hénon-Heiles (HH) system [26] that describes the non-linear motion of a star around a galactic center with the motion restricted to a plane. The HH system has four degrees of freedom in the phase space where
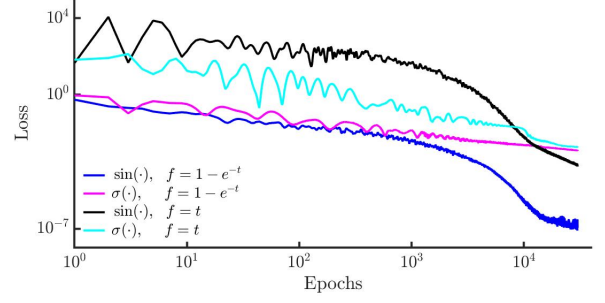

Fig. 4. NN solves the equations of motion for the HH system. Loss function in log-scale during the training for a different combinations of activation and parametric functions f shown by the legend.

$\mathbf{z} = (\mathbf{q}, \mathbf{p})^T = (x, y, p_x, p_y)^T$. The Hamiltonian and the total energy of this system is

$$H = \frac{1}{2} \left(p_x^2 + p_y^2 \right) + \frac{1}{2} \left(x^2 + y^2 \right) + \left(x^2 y - \frac{y^3}{3} \right). \quad (23)$$

The Hamilton's equations results in the nonlinear DEs system:

$$\dot{x} = p_x, \qquad \dot{y} = p_y, \qquad (24)$$
$$\dot{p}_x = - \left(x + 2xy \right), \qquad \dot{p}_y = - \left(y + x^2 - y^2 \right). \quad (25)$$

For the HH system we are seeking approximate solutions $\hat{\mathbf{z}} \in \mathbb{R}^4$. Accordingly, we employ a fully connected feed-forward NN with four outputs $\mathbf{N} \in \mathbb{R}^4$ used to parametrize $\hat{\mathbf{z}}$ according to the general formula (6). The initial conditions for the numerical experiment are $(x_0, y_0, p_{x,0}, p_{y,0}) = (0.3, -0.3, 0.3, 0.15)$, corresponding to the energy $E_0 = 0.13$. The maximal Lyapunov exponent for this set of initial conditions is $\lambda = 0.069$, and since λ is positive, the motion is chaotic [27]. The network consists of two hidden layers with 50 neurons per hidden layer. An equally spaced grid of $K = 100$ is initialized in the time interval $t = [0, 6\pi]$ that corresponds to 1.3 Lyapunov times. These points are used as the training set and are perturbed in the beginning of every epoch by using a random term obtained by a normal distribution with zero mean and with a standard deviation 0.18π. The loss function is defined by Eqs. (24), (25), and according to Eq. (7), as

$$L = \frac{1}{K} \sum_{n=0}^{K} \left[\left(\dot{\hat{x}}^{(n)} - \hat{p}_x^{(n)} \right)^2 + \left(\dot{\hat{y}}^{(n)} - \hat{p}_y^{(n)} \right)^2 \right.$$
$$+ \left(\dot{\hat{p}}_x^{(n)} + \hat{x}^{(n)} + 2\hat{x}^{(n)}\hat{y}^{(n)} \right)^2$$
$$\left. + \left(\dot{\hat{p}}_y^{(n)} + \hat{y}^{(n)} + \left(\hat{x}^{(n)} \right)^2 + \left(\hat{y}^{(n)} \right)^2 \right)^2 \right]. \quad (26)$$

We examine four different network architectures similar to the nonlinear oscillator system, namely for different activation and parametric functions. The networks are trained for $3 \cdot 10^4$ epochs by using Adam optimizer with learning rate $8 \cdot 10^{-3}$. After training for long enough to ensure convergence in the loss function we find this number of epochs is sufficient to optimize the network. In Fig. 4, we show the loss function (26) in the training where each color corresponds to a different
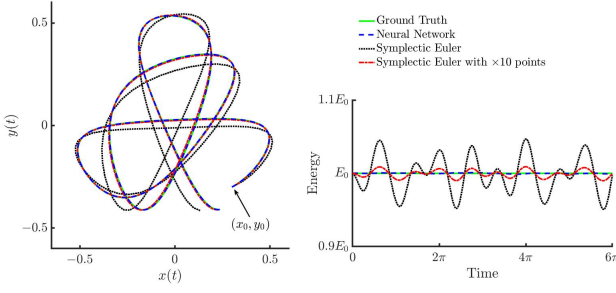
Fig. 5. Left: The orbit for the HH system in the $x - y$ plane obtained by a NN (blue) that is trained in $K = 100$ time points and by symplectic Euler integrator evaluated in K (green) and $10 \times K$ (orange) points. Red curves are considered as the ground truth and obtained by a numerical solver. Right: Energy of the HH system with time. The Hamiltonian NN conserves energy locally while the symplectic Euler method does not maintain constant energy levels even at the highest resolution.

architectures according to the legend in the figure. Again, the choice of $\sin(\cdot)$ activation and $f(t) = 1 - e^{-t}$ yields the best network performance. In Fig. 5, we compare the approximated trajectories and the energy obtained by the symplectic NN (blue lines), and by a symplectic Euler integrator which is evaluated in K and in $10 \times K$ time points (shown by black and red lines, respectively). Solutions obtained by a solver are considered as the ground truth (green curves). The left panel in Fig. 5 shows the orbit in the $x - y$ plane where the Hamiltonian NN solution is indistinguishable from the ground truth. The right panel represents the total energy in time where the NN solutions conserve the energy better than the solutions obtained by the symplectic Euler method. The symplectic Euler must use an order of magnitude higher resolution than NN to capture the correct orbit portrait, however, the energy is still not conserved locally. We find, but do not show, in Fig. 5 that the symplectic Euler requires two order of magnitude higher resolution in order to conserve the energy as well as the solutions obtained by the Hamiltonian network.

VI. Conclusion

In recent years, machine learning has made in-roads in traditional science and engineering fields. NNs have attracted scientists' interest due to their outstanding capabilities in regression, classification, and prediction tasks. Since these methods are relatively new to physics, there are many physical concepts that have not been embedded yet in the structure of NNs. In this work, we proposed a physics-inspired unsupervised NN for solving DEs that describe the spatio-temporal motion of dynamical systems. The Hamiltonian formulation is embedded in the NN through the loss function and therefore, the predicted solutions conserve energy. A smooth and bounded parametric form of solutions was introduced in this study that makes the proposed architecture a symplectic network, and subsequently, a time-invariant unit. By appropriately choosing the activation function a better domain knowledge is provided that drastically improves the network performance. Moreover, the proposed Hamiltonian architecture allows the network outputs to share their weights. Sharing the learning parameters helps the NN to discover underlying co-dependencies and

subsequently, improves the network predictability in learning solutions that satisfy nonlinear systems of DEs. An error analysis was developed in this work which can be used to analyze how the errors in the predicted solutions propagate in time. In addition, this error analysis provides a threshold in the loss function, where we can early-stop training the network when a certain accuracy occurs, namely a lower error in the predicted solutions is ensured.

There are several advantages in using NN solvers instead of traditional numerical integrators for solving DEs. The solutions obtained by a NN are continuous, smooth, and in an analytical form. Due to many outputs with shareable weights, the Hamiltonian NN discovers solutions that satisfy the Hamilton equations simultaneously and consistently. Subsequently, the NN solver conserves the correct Hamiltonian in contrary to symplectic integrators that conserve a slightly perturbed Hamiltonian. We outlined that the solutions obtained by the NN conserve the energy locally along with all the time points, and out-performs the symplectic Euler integrator that predicts an energy with a fluctuating error term. In problems where energy conservation is crucial, the Hamiltonian NN will show better performance than symplectic integrators. The Hamiltonian formulation provides a solid framework for theoretical extension in many areas of physics such perturbation approaches and theory of chaos, as well as statistical and quantum mechanics. Hence, the proposed Hamiltonian NN provides fertile ground on which modern research problems can potentially be handled.

References

[1] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, 2018.

[2] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks," *Proceeding of the Royal Sociaty A-Mathematical Physical and Engineering Sciences*, vol. 474, no. 2213, 2018.

[3] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," *Chaos*, vol. 27, no. 4, 2017.

[4] G. Neofotistos, M. Mattheakis, G. D. Barmparis, J. Hizanidis, G. P. Tsironis, and E. Kaxiras, "Machine Learning With Observers Predicts Complex Spatiotemporal Behavior," *Frontiers in Physics*, vol. 7, 2019.

[5] N. Mohajerin and S. L. Waslander, "Multistep Prediction of Dynamic Systems With Recurrent Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3370–3383, 2019.

[6] J. Ling, R. Jones, and J. Templeton, "Machine learning strategies for systems with invariance properties," *Journal of Computational Physics*, vol. 318, pp. 22–35, 2016.

[7] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[8] R. Fang, D. Sondak, P. Protopapas, and S. Succi, "Neural network models for the anisotropic reynolds stress tensor in turbulent channel flow," *Journal of Turbulence*, vol. 0, no. 0, pp. 1–19, 2019. [Online]. Available: https://doi.org/10.1080/14685248.2019.1706742

[9] K. Duraisamy, G. Iaccarino, and H. Xiao, "Turbulence Modeling in the Age of Data," *Annual Review of Fluid Mechanics*, vol. 51, pp. 357–377, 2019.

[10] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *Journal of Computational Physics*, vol. 335, pp. 736–746, 2017.

[11] ——, "Machine learning of linear differential equations using Gaussian processes," *Journal of Computational Physics*, vol. 348, pp. 683–693, 2017.

[12] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, 2017.

[13] J. N. Kutz, S. H. Rudy, A. Alla, and S. L. Brunton, "Data-driven discovery of governing physical laws and their parametric dependencies in engineering, physics and biology," *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pp. 1–5, 2017.

[14] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, no. 31, pp. 15 344–15 349, 2019.

[15] 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*, vol. 378, pp. 686–707, 2019.

[16] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, pp. 987–1000, 1998.

[17] J. A. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018.

[18] M. Magill, F. Qureshi, and H. W. de Haan, "Neural networks trained to solve differential equations learn general representations," in *NeurIPS*, 2018.

[19] J. Han, A. Jentzen, and E. Weinan, "Solving high-dimensional partial differential equations using deep learning." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 115 34, pp. 8505–8510, 2017.

[20] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

[21] E. Noether, "Invariante variationsprobleme," *Math. Phys. Klasse*, vol. 2, pp. 235 – 257, 1918.

[22] B. J. Leimkuhler and R. D. Skeel, "Symplectic numerical integrators in constrained hamiltonian systems," *Journal of Computational Physics*, vol. 112, no. 1, pp. 117–125, 1994.

[23] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. Devito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[25] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, 2007.

[26] M. Hénon and C. Heiles, "The applicability of the third integral of motion: Some numerical experiments," *The Astronomical Journal*, vol. 69, pp. 73–79, 1964.

[27] I. Shevchenko and A. Mel'Nikov, "Lyapunov exponents in the hénon-heiles problem," *JETP Letters*, vol. 77, no. 12, pp. 642–646, 12 2003.