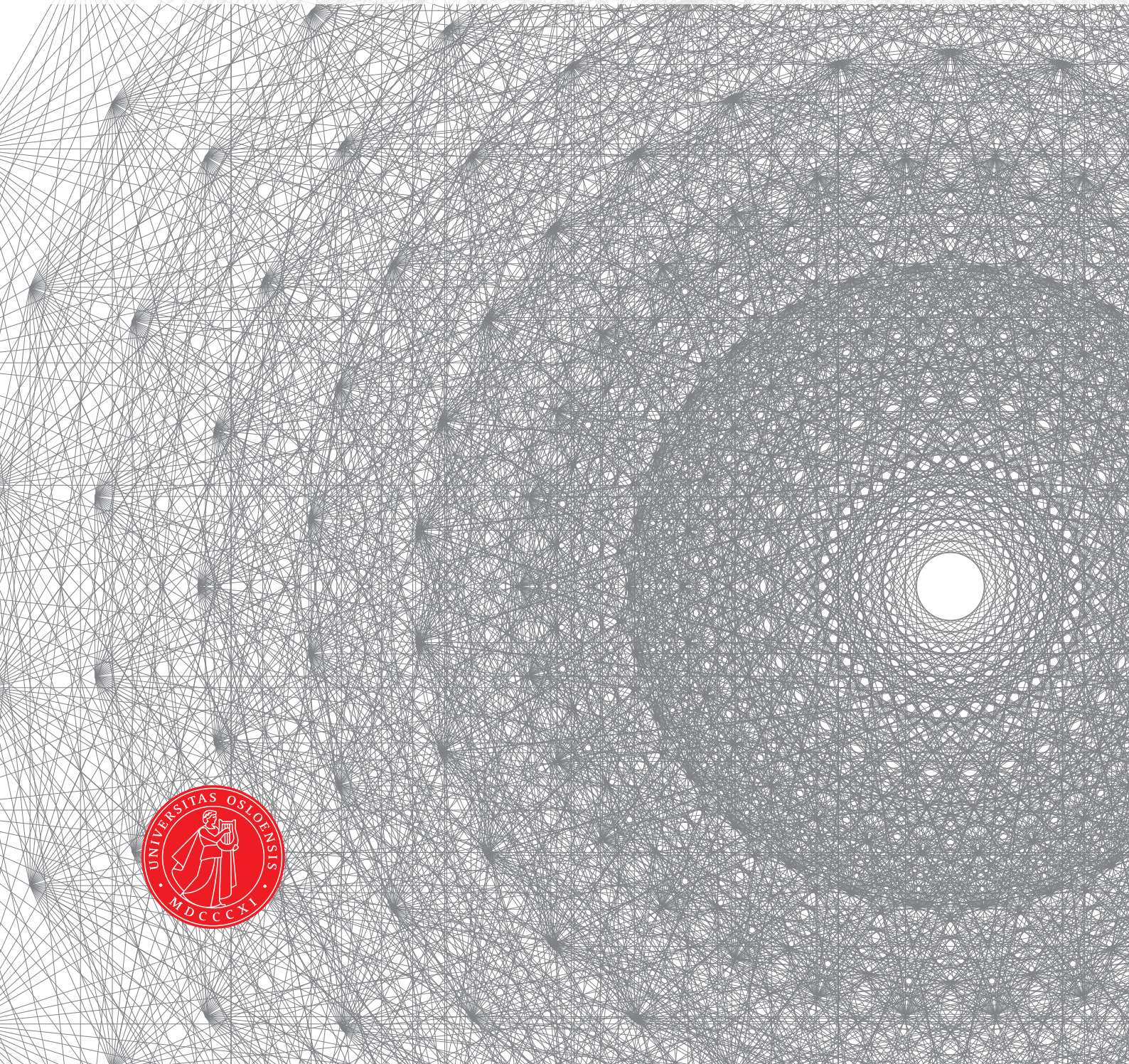
UiO : Department of Mathematics
University of Oslo

Pseudo-Hamiltonian System Identification

Sigurd Holmsen

Master's Thesis, Spring 2023



This master's thesis is submitted under the master's programme *Computational Science*, with programme option *Applied Mathematics and Risk Analysis*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Abstract

This thesis concerns the application of physics-informed machine learning to dynamical systems that can be represented as first-order ordinary differential equations. Current system identification models struggle to learn energy-preserving dynamical systems where damping and external forces affect the training data. We will tackle this problem by letting our model assume a pseudo-Hamiltonian structure, meaning we learn the inner and outer dynamics separately. We use system identification to learn the inner dynamics, while a neural network will generally be employed to learn the external forces. But, we also explore the possibility of learning the external forces through system identification. Furthermore, we introduce an integration scheme for training the model that attempts to handle noisy data.

Acknowledgements

First, I would like to thank all of my supervisors. Sølve Eidnes, you have given me continuous guidance and support. Your extensive knowledge of mechanics, a field I was not very familiar with, has been critical in helping me gain a better understanding of the subject matter. Vegard Antun and Øyvind Ryan, you have provided great insight, especially during the writing process. Your honest and constructive criticism has been invaluable in helping me to improve my work. Signe Riemer-Sørensen, you have helped me to strive for good results, and your ambition for my thesis has been an inspiration. You have all been motivating and fun to work with, and I am grateful for all of you.

Second, I would like to express appreciation for the fellow mathematics students I have met throughout the past five years. There are too many names to mention here, but I want to give a special shout-out to the members of my study group, Clopen[1]. Øystein Bruce, you have been a great student companion, and you are great at pointing out the details I miss. And David Andersen, your positivity and thoughtful insights have been appreciated. I hope we can continue to be a great team in the future. Lastly, I want to thank my friends, family, and girlfriend for being supportive and cheering me on during the last year.

Sigurd Holmsen,
Oslo 2023

[1] https://www.mn.uio.no/math/studier/aktuelt/aktuelle-saker/2020/maec-studenter-vant-nb-casenm.html

Contents

CHAPTER 1

Introduction

Part of the work in this master's thesis is found in a paper also named *Pseudo-Hamiltonian System Identification* [HER23] written by the author along with supervisors. The code used in this thesis is available on the author's GitHub page[1].

One of the goals of the natural sciences is to accurately describe real-life physical phenomena. Differential equations, invented along with calculus, are well suited to describe systems that change over time, and they have near-endless applications. The problem of describing physical systems through differential equations has historically been tackled in two ways: qualitative analysis, i.e. through physical knowledge or intuition, and quantitative analysis, i.e. analyzing physical data gathered from experiments. Physics-informed machine learning is the combination of these two approaches. We guide a machine learning model using prior physical knowledge so that this knowledge does not need to be relearned [Kar+21; RPK19; Wil+22]. This approach is especially relevant for learning dynamical systems. Given a first-order ordinary differential equation (ODE) of dimension n

$$\dot{x} = g(x, t), \quad x \in \mathbb{R}^n, \quad t \in [0, \infty), \tag{1.1}$$

we can approximate the right-hand side by learning from collected data in combination with assumptions from physics. Recently, several works have imposed the physical law of energy conservation in isolated systems on machine-learning models. Hamiltonian and Lagrangian mechanics are convenient frameworks for doing this [Che+20; Cra+20; GDY19]. More recently, [DA21] and [Eid+23] have made this approach more applicable to real-world data using the *pseudo-Hamiltonian* framework: a generalization of the Hamiltonian formulation that adds damping and external forces. In other words, the framework allows for a relaxation of the law of energy conservation. [Eid+23] learn these systems using *pseudo-Hamiltonian neural networks* (PHNNs), and the key innovation is their ability to learn the pure Hamiltonian dynamics, damping effects, and external forces using separate models. The example used to demonstrate this is a system of tanks connected by pipes where there is a leak in one of the tanks. Even if the PHNN model is trained on the leaky system, the PHNN model can predict future states after the leak is fixed. In other words, a model trained under suboptimal conditions is able to model the system under optimal

[1] https://github.com/sigurho/PHSI

conditions.

Although PHNNs can successfully produce accurate predictions, they are black-box models that do not reveal the true governing equations of the system. The field of system identification copes with this issue. The goal of this field is to extract information about a system's true governing equations from data. In the case of differential equations, the goal becomes to learn the true function g in Equation 1.1. Learning the analytical terms of the system holds two clear advantages over black-box models: First, analytic terms can give insight into the dynamical system that may enhance interpretation and stimulate further developments. Second, accurate analytical models are more resistant to the limited extrapolation capabilities that render many machine learning models unusable beyond the domain of the training data. System identification has been applied to learning both general dynamical systems [BL07; BPK16; SL09] and Hamiltonian systems [DXZ20], but not to pseudo-Hamiltonian systems.

In this thesis, we attempt to answer the following questions:

- Is it possible to learn an energy-preserving dynamical system using system identification when damping and external forces affect the training data?
- Is it also possible to learn the energy-preserving system, the damping, and the external forces all at once using separate system identification models?

To tackle these questions, we introduce a novel system identification model that assumes a pseudo-Hamiltonian structure. We call it *pseudo-Hamiltonian system identification* (PHSI). It models the pure Hamiltonian and damping components using system identification and generally models the external forces using a neural network without prior knowledge of how these components are separated. In some cases, system identification can also be applied to learning the external forces. In other words, the PHSI model combines the general pseudo-Hamiltonian structure that allows for the imposing of physical knowledge with the white-box system identification approach that gives insight and interpretability. Thus, a major limitation of system identification will be addressed: many frameworks encounter difficulty in learning a dynamical system from sampled data that is suboptimal, such as when an external disturbance affects the dynamics. We also gain a deeper insight into pseudo-Hamiltonian systems than what is possible with the black-box PHNNs of [Eid+23]. Furthermore, we address the problem of learning a system described by differential equations when data on the derivatives are unavailable by proposing a training scheme inspired by numerical integration. This has been discussed for neural network models by [Jin+20; MIY20; Zhu+22], but less so for system identification models. In this integration scheme, we elaborate on why symmetric integrators are the best choice for handling noisy data.

In summary, the goal of this thesis is to advance the use of system identification by applying it to realistic dynamical systems where the dynamics are disturbed by external forces. If we are successful, we will have an advantage over black-box learning models when it comes to interpretability and predictive capability.

PART I

Background

CHAPTER 2

Hamiltonian mechanics

Hamiltonian mechanics provides a convenient framework for imposing the physical law of energy conservation on the machine-learning models that will be introduced later in chapters 3 and 4. Hamiltonian mechanics was introduced in the early 1800s by William Rowan Hamilton and is a reformulation of classical mechanics. It describes the evolution of a physical system in terms of the total energy of the system, rather than using the Lagrangian formulation that describes the system in terms of the action. It provides a convenient framework for analyzing systems with constraints. To understand the derivation of Hamiltonian mechanics, we briefly introduce Lagrangian mechanics and how it relates to the Hamiltonian framework.

2.1 Lagrangian mechanics

Lagrangian mechanics is a formulation of classical mechanics that describes the motion of a system using a scalar function called the Lagrangian. The following description will follow along the lines of [MR99] (pg. 1-5). Assume we want to study a physical system of d dimensions where the coordinates of the system are described as a function of time:

$$q(t) = (q_1(t), ..., q_d(t)),\ q : [0, \infty) \to \mathbb{R}^d \tag{2.1}$$

where $t > 0$ is the time variable. We denote the *system velocity* as the system coordinates differentiated with respect to time:

$$\dot{q}(t) = (\dot{q}_1(t), ..., \dot{q}_d(t)) = \left(\frac{dq_1(t)}{dt}, ..., \frac{dq_d(t)}{dt} \right). \tag{2.2}$$

For simplicity, we notate the system coordinates as $q = (q_1, ..., q_d)$, $\dot{q} = (\dot{q}_1, ..., \dot{q}_d)$ where $q, \dot{q} \in \mathbb{R}^d$. We let the *Lagrangian* be a smooth function $L : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

$$L(q, \dot{q}) = L(q_1, ..., q_d,\ \dot{q}_1, ..., \dot{q}_d), \tag{2.3}$$

which often represents the kinetic minus the potential energy in the system. Now fix two points $A, B \in \mathbb{R}^d$ and two time points $b > a > 0$. We introduce the functional known as the *action integral*

$$\mathcal{S}(q) = \int_a^b L(q, \dot{q}) dt, \tag{2.4}$$

defined for functions $q = (q_1, ..., q_d)$ belonging to the following class:

$$\mathcal{A} := \{q \in C^2([a,b] : \mathbb{R}^d) | q(a) = A, q(b) = B\}. \tag{2.5}$$

Hamilton's principle states that the true evolution of the system minimizes the action integral. That means we are interested in finding $x \in \mathcal{A}$ such that

$$\mathcal{S}(x) = \min_{q \in \mathcal{A}} \mathcal{S}(q). \tag{2.6}$$

We assume that this x exists. A good candidate for x is stationary points of $\mathcal{S}$, and it turns out that these stationary points can be found by solving a set of equations called the *Euler-Lagrange equations*. These will be derived in the following subsection.

The Euler-Lagrange equations

Assume we have a Lagrangian $L : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, two fixed points $A, B \in \mathbb{R}^d$, two time-points $b > a > 0$ and an action integral $\mathcal{S}(\cdot)$ given by Equation 2.4 and defined for functions $q \in \mathcal{A}$. We are interested in the function $x \in \mathcal{A}$ that minimizes the action integral, i.e. that satisfies Equation 2.6. A good candidate for this x is a stationary point of $\mathcal{S}$. Assume such a stationary point $y \in \mathcal{A}$ exists (the properties of a stationary point will be defined below) and note that y is not necessarily the minimizer x in Equation 2.6. Consider the perturbation $g_\varepsilon \in \mathcal{A}$ of y

$$g_\varepsilon(t) := (y_1(t) + \varepsilon_1 \eta_1(t), ..., y_d(t) + \varepsilon_d \eta_d(t)) \tag{2.7}$$

where $\eta_i(\cdot) \in C^2([a,b] : \mathbb{R})$ is an arbitrary differentiable function and $0 < \varepsilon_i \ll 1$ for $i = 1, ..., d$. We also impose the boundary condition $\eta_i(a) = \eta_i(b) = 0$ for $i = 1, ..., d$ which implies that $g_\varepsilon(a) = A,\ g_\varepsilon(b) = B$ is fulfilled. Then, we define

$$\dot{g}_\varepsilon(t) := (\dot{y}_1(t) + \varepsilon_i \dot{\eta}_1(t), ..., \dot{y}_d(t) + \varepsilon_d \dot{\eta}_d(t)). \tag{2.8}$$

Using $g_\varepsilon, \dot{g}_\varepsilon$, we can calculate the derivative of $\mathcal{S}$ with respect to a change in y_i, i.e.

$$\frac{d\mathcal{S}(g_\varepsilon)}{d\varepsilon_i} = \frac{d}{d\varepsilon_i} \int_a^b L(y_1 + \varepsilon_1 \eta_1, ..., y_d + \varepsilon_d \eta_d, \dot{y}_1 + \varepsilon_1 \dot{\eta}_1, ..., \dot{y}_d + \varepsilon_d \dot{\eta}_d)\, dt. \tag{2.9}$$

Since y is a stationary point of $\mathcal{S}$, we know that by definition that the derivative evaluated at $\varepsilon_i = 0$ is zero for all i, i.e.

$$\left.\frac{d\mathcal{S}(g_\varepsilon)}{d\varepsilon_i}\right|_{\varepsilon_i = 0} = 0, \quad i = 1, ..., d. \tag{2.10}$$

Using the chain rule, we calculate the left-hand side further for $i = 1, ..., d$:

$$\frac{d}{d\varepsilon_i}\bigg|_{\varepsilon_i=0} \int_a^b L(g_\varepsilon, \dot{g}_\varepsilon)\, dt = 0 \quad (2.11)$$

$$\int_a^b \frac{d}{d\varepsilon_i}\bigg|_{\varepsilon_i=0} L(y_1 + \varepsilon_1\eta_1, ..., y_d + \varepsilon_d\eta_d, \dot{y}_1 + \varepsilon_1\dot{\eta}_1, ..., \dot{y}_d + \varepsilon_d\dot{\eta}_d)\, dt = 0 \quad (2.12)$$

$$\int_a^b \left[\frac{dL}{d(y_i + \varepsilon_i\eta_i)}\frac{d(y_i + \varepsilon_i\eta_i)}{d\varepsilon_i} + \frac{dL}{d(\dot{y}_i + \varepsilon_i\dot{\eta}_i)}\frac{d(\dot{y}_i + \varepsilon_i\dot{\eta}_i)}{d\varepsilon_i}\right]\bigg|_{\varepsilon_i=0} dt = 0 \quad (2.13)$$

$$\int_a^b \left[\frac{dL}{d(y_i + \varepsilon_i\eta_i)}\eta_i + \frac{dL}{d(\dot{y}_i + \varepsilon_i\dot{\eta}_i)}\dot{\eta}_i\right]\bigg|_{\varepsilon_i=0} dt = 0 \quad (2.14)$$

$$\int_a^b \left[\frac{dL}{dy_i}\eta_i + \frac{dL}{d\dot{y}_i}\dot{\eta}_i\right] dt = 0. \quad (2.15)$$

Using integration by parts, we get the following

$$\int_a^b \left[\frac{\partial L}{\partial y_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}_i}\right]\eta_i\, dt + \left[\eta_i \frac{\partial L}{\partial \dot{y}_i}\right]_a^b = 0, \quad (2.16)$$

and then we insert the boundary conditions $\eta_i(a) = \eta_i(b) = 0$ to get

$$\int_a^b \left[\frac{\partial L}{\partial y_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}_i}\right]\eta_i\, dt = 0. \quad (2.17)$$

Here, we apply the fundamental lemma of calculus of variations [JJL98]. The lemma states that if this equation is to hold for all compactly supported smooth functions η_i, then $\frac{\partial L}{\partial y_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}_i}$ is identically zero. Note that η_i is compactly supported: since $\text{supp}(\eta_i) \in [a, b]$, the closure of $\text{supp}(\eta_i)$ is both closed and bounded meaning it is compact. Hence we get the desired *Euler-Lagrange* equations

$$\frac{\partial L}{\partial y_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}_i} = 0, \quad i = 1, ..., d. \quad (2.18)$$

Since $\eta_i,\ i = 1, ..., d$, is arbitrary (apart from being zero at the endpoints), Equation 2.10 and the Euler-Lagrange equations (Equation 2.18) are equivalent.

Note that if any $y \in \mathcal{A}$ satisfies the Euler-Lagrange equations, it only implies that it is a stationary point of the action integral $\mathcal{S}$, not that it is the minimizer of $\mathcal{S}$. However, since a minimizer is a stationary point, any minimizer $x \in \mathcal{A}$ will solve the Euler-Lagrange equations, as proved in [Eva22], page 116.

Example

We regard a system of N particles moving in Euclidian 3-space. The state of the system is described by $(q_1, ..., q_N)$ where $q_i : [0, \infty) \to \mathbb{R}^3,\ i = 1, ..., N$. L is the total kinetic minus potential energy in the system

$$L(q, \dot{q}) = \frac{1}{2}\sum_{i=1}^{N} m_i \|\dot{q}_i\|_2^2 - V(q_i)$$

where m_i and $V(q_i)$ are the mass and potential energy of particle q_i, respectively. In this case, the Euler-Lagrange equations (Equation 2.18) reduce to Newton's second law

$$\frac{d}{dt}(m_i\dot{q}_i) = -\frac{\partial V}{\partial q_i};\ i = 1, ..., N$$

i.e. force equals mass times acceleration for moving particles in a potential field.

2.2 Hamilton's equations

Keeping the definition of the Lagrangian in mind, we move on to the Hamiltonian formalism. We will transform the Euler-Lagrange equations, a system of d second-order ODEs, into Hamilton's equations, a system of $2d$ first-order ODEs. In this section, we assume that $q \in \mathcal{A}$ satisfies Euler-Lagrange equations (Equation 2.18). First, introduce *generalized momentum*

$$p(t) = (p_1(t), ..., p_d(t)) := \left(\frac{\partial L(q(t), \dot{q}(t))}{\partial \dot{q}_1}, ..., \frac{\partial L(q(t), \dot{q}(t))}{\partial \dot{q}_d}\right), \tag{2.19}$$

and we make the change of variables $(q, \dot{q}) \mapsto (q, p)$. Then, we introduce the *Hamiltonian*

$$H(q, p) = \sum_{i=1}^{d} p_i\dot{q}_j - L(q, p). \tag{2.20}$$

Using the chain rule, we make the following computations:

$$\frac{\partial H}{\partial p_i} = \dot{q}_i + \sum_{j=1}^{d}\left(p_j\frac{\partial \dot{q}_j}{\partial p_i} - \frac{\partial L}{\partial \dot{q}_j}\frac{\partial \dot{q}_j}{\partial p_i}\right) = \dot{q}_i + \sum_{j=1}^{d}\frac{\partial \dot{q}_j}{\partial p_i}\left(p_j - \frac{\partial L}{\partial \dot{q}_j}\right) = \dot{q}_i \tag{2.21}$$

and

$$\frac{\partial H}{\partial q_i} = \sum_{j=1}^{d} p_j\frac{\partial \dot{q}_j}{\partial q_i} - \frac{\partial L}{\partial q_i} - \sum_{j=1}^{d}\frac{\partial L}{\partial \dot{q}_j}\frac{\partial \dot{q}_j}{\partial q_i} = -\frac{\partial L}{\partial q_i}, \tag{2.22}$$

where Equation 2.19 has been used twice. Using the equations 2.18 and 2.19, we can rewrite Equation 2.22 as

$$\frac{\partial H}{\partial q_i} = -\frac{d}{dt}p_i \tag{2.23}$$

Thus, we have used the Euler-Lagrange equations to arrive at *Hamilton's equations*

$$\frac{\partial H}{\partial q_i} = -\dot{p}_i \quad , \quad \frac{\partial H}{\partial p_i} = \dot{q}_i \tag{2.24}$$

for $i = 1, ..., d$.

Proposition 2.2.1. *Let $H : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a Hamiltonian that is not explicitly time-dependent and that satisfies Hamilton's equations (Equation 2.24). Then, the Hamiltonian does not vary with time, i.e. $\frac{dH}{dt} = 0$.*

Proof:

$$\frac{dH}{dt} = \sum_{i=1}^{d} \frac{\partial H}{\partial p_i}\frac{dp_i}{dt} + \sum_{i=1}^{d} \frac{\partial H}{\partial q_i}\frac{dq_i}{dt}$$
$$= \sum_{i=1}^{d} \frac{\partial H}{\partial p_i}\left(-\frac{\partial H}{\partial q_i}\right) + \sum_{i=1}^{d} \frac{\partial H}{\partial q_i}\frac{\partial H}{\partial p_i} = 0$$

2.3 Hamiltonian systems

A Hamiltonian system is an ODE that can be expressed through gradients of a Hamiltonian function that is preserved over time. In Section 2.2, the Hamiltonian was described as a function H taking the variables $q, p \in \mathbb{R}^d$ as input (see Equation 2.20), and it was shown that this function satisfies Hamilton's equations (see Equation 2.24). This Hamiltonian can be used to express the ODE

$$\begin{pmatrix} \dot{q} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix} \begin{pmatrix} \partial H/\partial q \\ \partial H/\partial p \end{pmatrix}, \tag{2.25}$$

where I_d is the identity matrix and H is preserved in time as shown in Proposition 2.2.1. Notice that since the system in Equation 2.25 is of $2d$ dimensions, it can only describe ODEs that have an even number of dimensions. We define a more general class of Hamiltonian systems that describe ODEs of n dimensions where n can be either even or odd. Given a system described by the coordinates $x \in \mathbb{R}^n$, a Hamiltonian system can be most generally described as

$$\dot{x} = S(x)\nabla H(x), \tag{2.26}$$

where $x \in C^2([0, \infty) : \mathbb{R}^n)$ denotes the coordinates of the system as function of time $t > 0$, the *structure matrix* $S : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is such that $S(x)$ is an antisymmetric matrix for any x, and $H : \mathbb{R}^n \to \mathbb{R}$ is the Hamiltonian. Equation 2.26 can be considered a generalization of Equation 2.25, and notice that n can be either even or odd. In this more general definition, H will still be preserved in time following the anti-symmetry of S similar to what was shown in Proposition 2.2.1:

$$\dot{H} = \nabla H^T \begin{pmatrix} \dot{q} \\ \dot{p} \end{pmatrix} = \nabla H^T \, S \, \nabla H = 0. \tag{2.27}$$

A Hamiltonian system on the form in Equation 2.25 is a special case of Equation 2.26 where $x = (q, p)$ and $S = \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix}$. This special case is known as the *canonical* formulation of a Hamiltonian system. If a system can be expressed by Equation 2.26 and not Equation 2.25, it is known as a Hamiltonian system of *non-canonical* formulation. An ODE system can be expressed as in Equation (2.26) by several non-unique pairs of H's and S's [MQR99]. This is explored further in Section 2.5.

2.4 Existence of the Hamiltonian formulation

Equation 2.27 shows that the Hamiltonian structure guarantees the preservation of a Hamiltonian H in the Hamiltonian formulation (see Equation 2.26). [MQR99] prove the reverse: given that a function H is preserved for the solution of an ODE $\dot{x} = g(x)$, there will always exist a structure matrix S ensuring that a Hamiltonian formulation can be made. A simplified version of the proof is given:

Proposition 2.4.1. *Assume we have an ODE $\dot{x} = g(x)$ of where $x \in C^2([0,\infty) : \mathbb{R}^n)$, and that there exists a function $H : \mathbb{R}^n \to \mathbb{R}$ such that $H(x) = H(x_0)$ for all $\{x|\dot{x} = g(x)\}$ given the initial condition $x_0 = x(0)$, meaning the Hamiltonian value is preserved for all solutions of the system. Then, there exists a matrix $S : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ such that the ODE can be rewritten in the Hamiltonian formulation:*

$$\dot{x} = S(x)\nabla H(x).$$

Proof: To prove that the system can be written in the Hamiltonian form, we have to prove the existence of the structure matrix S. S has two properties that must be fulfilled: anti-symmetry and that it maps ∇H to g. Thus, we seek to construct a matrix that meets these two requirements expressed in terms of g and H. Assume that ∇H is non-vanishing. The matrix $\frac{1}{|\nabla H|^2} g(\nabla H)^T$ maps ∇H to g because $\frac{1}{|\nabla H|^2} g(\nabla H)^T(\nabla H) = g$. Since H is conserved in time (see Equation 2.27),

$$0 = \dot{H} = \dot{x}^T \nabla H = g^T \nabla H = 0. \tag{2.28}$$

In other words, the term $\nabla H f^T$ maps ∇H to 0. We construct:

$$S = \frac{1}{|\nabla H|^2}\left(g(\nabla H)^T - (\nabla H)g^T\right) \tag{2.29}$$

which both maps ∇H to g and is antisymmetric because

$$S = \frac{1}{|\nabla H|^2}\left(g(\nabla H)^T - (\nabla H)g^T\right) = -\frac{1}{|\nabla H|^2}\left(g(\nabla H)^T - (\nabla H)g^T\right)^T = -S^T$$

Hence, using the constructed, anti-symmetric S, we can express the ODE as

$$\dot{x} = S(x)\nabla H(x). \quad \blacksquare$$

The constructed matrix S may or may not be dependent on the input x, and may or may not be of canonical structure (see Equation 2.25). [MQR99] also prove that S is bounded in the neighborhood of any non-degenerate stationary point of H. It is also important to notice that this is just one particular matrix that meets the requirements of S in Equation 2.26 and that other such matrices may exist given the same ODE and Hamiltonian. In other words, the Hamiltonian formulation is generally not unique.

Example

As an example of the construction of the matrix S, we look at the mass-spring system. The motion of the system is described through the following ODE:

$$g(x) = \begin{pmatrix} \dot{q} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} p/m \\ -kq \end{pmatrix} = \begin{pmatrix} p \\ -q \end{pmatrix} \tag{2.30}$$

where $x = (q, p)^T$, and we set the mass $m = 1$ and the spring's force coefficient $k = 1$. The system has a related Hamiltonian function

$$H(q, p) = \frac{1}{2}\left(q^2 + p^2\right) \tag{2.31}$$

and the following gradient

$$\nabla H = \begin{pmatrix} q \\ p \end{pmatrix}.$$

that represents the energy in the system. From this, we can construct the matrix S using Equation 2.29:

$$\begin{aligned} S(q,p) &= \frac{1}{q^2+p^2}\left(\begin{pmatrix} p \\ -q \end{pmatrix}(q,p) - \begin{pmatrix} q \\ p \end{pmatrix}(p,-q)\right) \\ &= \frac{1}{q^2+p^2}\left(\begin{pmatrix} qp & p^2 \\ -q^2 & -qp \end{pmatrix} - \begin{pmatrix} qp & -q^2 \\ p^2 & -qp \end{pmatrix}\right) \\ &= \frac{1}{q^2+p^2}\begin{pmatrix} 0 & q^2+p^2 \\ -q^2-p^2 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \end{aligned}$$

In this case, the matrix S computed using Equation 2.29 turned out to give the canonical formulation. However, there is in general no guarantee of computing the canonical formulation using Equation 2.29, even if such a formulation exists for a given ODE.

2.5 Non-uniqueness of the Hamiltonian formulation

The Hamiltonian formulation of an ODE as $\dot{x} = S(x)\nabla H(x)$ is in general not unique. To show this, we create a more general way of constructing the S matrix in Equation 2.29. Assume we have an ODE $\dot{x} = g(x)$ where $x \in C^2([0, \infty) : \mathbb{R}^n)$ which has a related Hamiltonian function H such that $H(x) = H(x_0)$ for all $\{x|\dot{x} = g(x)\}$ given the initial condition $x_0 = x(0)$. For any non-vanishing function y, we can construct an $n \times n$ matrix

$$S_y(x) = \frac{1}{y(x)^T \nabla H(x)}(g(x)y(x)^T) - y(x)g(x)^T) \tag{2.32}$$

that is anti-symmetric. Since $g^T \nabla H = \dot{x}^T \nabla H = \frac{dx}{dt}\frac{\partial H}{\partial x} = \frac{dH}{dt} = 0$, we get

$$S_y(x)\nabla H(x) = \frac{1}{y(x)^T \nabla H(x)}(g(x)y(x)^T \nabla H(x)) - y(x)g(x)^T \nabla H(x)) \tag{2.33}$$

$$= \frac{1}{y(x)^T \nabla H(x)}(g(x)y(x)^T)\nabla H(x)) = g(x), \tag{2.34}$$

meaning S_y is both antisymmetric and maps ∇H to g, and thus the constructed S_y has the required properties in the canonical Hamiltonian formulation (see Equation 2.26). Now, since the choice of y is non-unique, the constructed matrix S_y and followingly the Hamiltonian formulation is non-unique.

Example

To further illustrate the non-uniqueness of the Hamiltonian formulation, we study the Kepler problem as described in [DOY11]. The system is given by the four-dimensional ODE

$$\dot{x}_1 = x_3, \quad \dot{x}_2 = x_4, \quad \dot{x}_3 = -\frac{x_1}{(x_1^2 + x_2^2)^{3/2}}, \quad \dot{x}_4 = -\frac{x_2}{(x_1^2 + x_2^2)^{3/2}}. \tag{2.35}$$

This system preserves multiple Hamiltonians: the canonical Hamiltonian

$$H_1(x) = \frac{1}{2}(x_3^2 + x_4^2) - \frac{1}{\sqrt{x_1^2 + x_2^2}}, \tag{2.36}$$

the angular momentum

$$H_2(x) = y_1 y_4 - y_2 y_3, \tag{2.37}$$

and the *Runge-Lenz* vector

$$H_3(x) = x_2 x_3^2 - x_1 x_3 x_4 - \frac{x_2}{\sqrt{x_1^2 + x_2^2}}, \tag{2.38}$$

$$H_4(x) = x_1 x_4^2 - x_2 x_3 x_4 - \frac{x_1}{\sqrt{x_1^2 + x_2^2}}. \tag{2.39}$$

By Equation 2.29, every Hamiltonian H_1, H_2, H_3, and H_4 has a corresponding antisymmetric matrix S_1, S_2, S_3, and S_4 that can be used to write the ODE in the form of Equation 2.26, i.e. the ODE can be expressed by each of the Hamiltonians as $\dot{x} = S_i(x)\nabla H_i(x),\ i = 1, 2, 3, 4$. This shows that some systems can preserve multiple Hamiltonian functions, resulting in multiple distinct formulations.

This system can also exemplify the non-uniqueness of the structure matrix for a fixed Hamiltonian. We formulate a Hamiltonian formulation using H_1 in Equation 2.36. Firstly, H_1 can be used to express the canonical formulation:

$$\dot{x} = \begin{pmatrix} 0 & I_2 \\ -I_2 & 0 \end{pmatrix} \nabla H_1(x)$$

If we now insert H_1 and g into Equation 2.29, we get the following matrix:

$$S_1(x) = \frac{1}{\frac{1}{(x_1^2+x_2^2)^2} + x_3^2 + x_4^2} \cdot$$

$$\begin{pmatrix} 0 & \frac{x_2x_3 - x_1x_4}{(x_1^2+x_2^2)^{3/2}} & x_3^2 + \frac{x_1^2}{(x_1^2+x_2^2)^3} & x_3x_4 + \frac{x_1x_2}{(x_1^2+x_2^2)^3} \\ \frac{x_1x_4 - x_2x_3}{(x_1^2+x_2^2)^{3/2}} & 0 & x_3x_4 + \frac{x_1x_4}{(x_1^2+x_2^2)^3} & x_4^2 + \frac{x_2^2}{(x_1^2+x_2^2)^3} \\ -x_3^2 - \frac{x_1^2}{(x_1^2+x_2^2)^3} & -x_3x_4 - \frac{x_1x_2}{(x_1^2+x_2^2)^3} & 0 & \frac{x_2x_3 - x_1x_4}{(x_1^2+x_2^2)^{3/2}} \\ -x_3x_4 - \frac{x_1x_2}{(x_1^2+x_2^2)^3} & -x_4^2 - \frac{x_2^2}{(x_1^2+x_2^2)^3} & \frac{x_1x_4 - x_2x_3}{(x_1^2+x_2^2)^{3/2}} & 0 \end{pmatrix}.$$

This matrix can also describe the exact same ODE using the exact same Hamiltonian H_1, i.e.

$$\dot{x} = S_1(x)\nabla H_1(x). \tag{2.40}$$

Thus, there is no guarantee of uniqueness in the Hamiltonian formulation, even when the Hamiltonian function is fixed. Notice that the canonical formulation is always unique in that there only exists one combination of structure matrix and Hamiltonian that can express a Hamiltonian system in a canonical formulation.

2.6 Separable Hamiltonians

In general, the Hamiltonian function is regarded as the total energy of the system. For some systems, the Hamiltonian can be split into two separate functions that can be interpreted as kinetic and potential energy, respectively. For this to be possible, the Hamiltonian has to be *separable*.

Definition 2.6.1. Assume a Hamiltonian system of canonical form (see Equation 2.25) with a Hamiltonian $H : \mathbb{R}^{2d} : \mathbb{R}$. H is *separable* if there exist functions $V, T : \mathbb{R}^d \to \mathbb{R}$ such that

$$H(q,p) = V(q) + T(p) \text{ for all } q, p \in \mathbb{R}^d. \tag{2.41}$$

If it is not possible to separate a Hamiltonian, it is simply called *nonseparable*.

2.7 Pseudo-Hamiltonian Systems

One of the strengths of the Hamiltonian formulation is the conservation of energy. However, in a more realistic real-world setting, there are no guarantees that a dynamical system conserves energy over time as it may not be completely isolated. We seek a generalization of the Hamiltonian formulation that loosens this constraint. Thus, introduce damping that allows for energy dissipation and external forces that describe disturbances in the system. We consider pseudo-Hamiltonian systems as described in [Eid+23]:

$$\dot{x} = (S(x) - R(x))\,\nabla H(x) \;+\; F(x,t), \tag{2.42}$$

where $R : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ is such that $R(x)$ is a positive semi-definite matrix describing the dissipation of x, and $F : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ denotes the external forces. We do not impose any restraints on the external forces F to make the formulation as general as possible. If we imposed certain conditions on F, this would be equivalent to the *port-Hamiltonian* formulation from control theory [Van06; VJ14]. The formulation is completely general and non-unique; any ODE can be represented and there are generally many equivalent formulations. As an example, for an ODE: $\dot{x} = g(x,t)$, we can set $S = R = 0$, and $F = g$, and the ODE is recovered. In other words, although we wish to have no restraints on F to make the model as general as possible, this allows for a large number of equivalent descriptions of the same system.

Because of its ability to describe damping and external forces, the pseudo-Hamiltonian formulation can describe more general and realistic models than a pure Hamiltonian formulation can. Even if a system is completely isolated,

we may want to study a subsystem for which energy is no longer preserved, and the exchange of energy within the system can be described as the external forces F.

CHAPTER 3

Supervised learning and neural networks

Machine learning, the idea that a machine can learn an algorithm or model from data, has existed for over half a century. The goal of machine learning can be general and broad, but typically, the goal is to convert experience into knowledge by feeding data into a learning algorithm. During the past decades, much progress has been made in applying machine learning models to fields including image analysis and speech recognition. In more recent times, machine learning has been applied to analyzing and learning physical systems from data. In this section, we review the basis of supervised learning and neural networks. We also discuss physics-informed machine learning.

3.1 Supervised learning

This summary follows along the lines of [Fou22]. In the supervised learning setting, we have access to data of the form

$$y_i = f(x_i) + \varepsilon_i, \quad i \in [1:n] \tag{3.1}$$

where the *instances* or *input data* $x_i \in \mathcal{X}$ and *targets* or *output data* $y_i \in \mathcal{Y}$ are known. $\mathcal{X} \subseteq \mathbb{R}^d$ is made up of vectors containing d different *features*. In the learning scenario known as *regression*, $\mathcal{Y} = \mathbb{R}^k$, and elements of $\mathcal{Y}$ are called *labels*. For $i = 1, ..., n$, the instance x_i is mapped to the target y_i by an unknown function $f : \mathcal{X} \to \mathcal{Y}$ with the addition of normally distributed noise ε_i, i.e. $\varepsilon_i \sim N(0, \sigma^2)$. $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$ is known as the *training data*. Using the training data, the objective is to produce or *train* a function $\hat{f}_\mathcal{S} : \mathcal{X} \to \mathcal{Y}$ called a *model* or *predictor* that approximates f. Given the input data $x \in \mathcal{X}$, we call $\hat{f}_\mathcal{S}(x)$ a *prediction*.

We evaluate a model $\hat{f}$ by how large the *risk* or *generalization error* is. This is defined as the expected loss

$$\text{Risk}_f(\hat{f}) = \mathbb{E}[\text{Loss}(\hat{f}(x), f(x))], \tag{3.2}$$

where the expectation is taken over the random variable $x \in \mathcal{X}$ whose distribution is unknown. The loss function $\text{Loss} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is, in general, a function that returns small values for pairs of output y and prediction $\hat{y}$ that

are close, and large values for pairs that are far apart. For regression problems, a common loss function is the square error:

$$\mathrm{SE}(\hat{y}, y) = \|\hat{y} - y\|_2^2. \tag{3.3}$$

Since we do not have access to the distribution of data sampled from $\mathcal{X}$, we cannot compute Equation 3.2. Therefore we approximate it by using the available data and the *mean square error* function

$$\mathrm{MSE}_{\mathcal{S}}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} \|\hat{f}(x_i) - y_i\|_2^2. \tag{3.4}$$

Since this thesis tackles regression problems, the risk will always be approximated by this function. When we refer to *training* a model, it means *empirical risk minimization*. Given a class of functions, or a *hypothesis class* $\mathcal{H} \subseteq \{f | f : \mathcal{X} \to \mathcal{Y}\}$, the empirical risk minimization means finding the function $\hat{f}$ that minimizes the empirical risk

$$\Delta_{\mathcal{H}}^{\mathrm{erm}} : \mathcal{S} \mapsto \underset{\hat{f} \in \mathcal{H}}{\operatorname{argmin}}\ \mathrm{MSE}_{\mathcal{S}}(\hat{f}). \tag{3.5}$$

We have to be careful when evaluating a trained model; a model with a low MSE score can still have high risk. This happens if the empirical risk minimization $\Delta_{\mathcal{H}}^{\mathrm{erm}}(\mathcal{S})$ learns the local noise in the dataset it is trained on $(\mathcal{S})$ and does not perform well on unseen data $x \in \{\mathcal{X} \setminus \{x_1, ..., x_n\}\}$. This is known as *overfitting*, and can happen if the hypothesis class $\mathcal{H}$ is too large. On the other hand, if $\mathcal{H}$ is too small, $\Delta_{\mathcal{H}}^{\mathrm{erm}}(\mathcal{S})$ is not flexible enough to achieve low empirical risk, something known as *underfitting*. Since $\Delta_{\mathcal{H}}^{\mathrm{erm}}(\mathcal{S})$ is designed to minimize empirical risk, the mean square error is not a reliable evaluation of the true risk of the model. To better estimate the true risk and evade overfitting, we split our dataset $\mathcal{S}$ into a training set $\mathcal{T}$ used to train the model $\hat{y} = \Delta_{\mathcal{H}}^{\mathrm{erm}}(\mathcal{T})$ and a validation set $\mathcal{V}$ used to calculate the empirical risk $\mathrm{MSE}_{\mathcal{V}}(\hat{y})$. Usually, $\mathcal{T}$ contains the majority of the dataset, e.g. 80%.

3.2 Neural networks

Neural networks are a vastly popular class of machine learning models that are inspired by the biology of an animal brain. One of its strengths is its ability to fit a wide variety of mathematical functions. There are many types of neural networks, so giving a precise definition that fits them all is difficult. In this thesis, we only consider a subclass of neural networks called *dense feedforward neural networks*. Before we go over the structure of the neural network, also called the *architecture*, we briefly explain some terminology:

- Neuron: a mathematical function that receives input vector $\{x_i\}_{i=1}^{d}$ and computes

  $$z(x) = g\left(\sum_{i=1}^{d} x_i w_i + b\right)$$

  where $w_i \in \mathbb{R},\ i = 1, ..., d$ and $b \in \mathbb{R}$ are weight and bias, respectively, and $g : \mathbb{R} \to \mathbb{R}$ is an activation function. The weights and the bias are trainable.

- Activation function: a non-linear, differentiable function that serves the purpose of reducing weak signals (in neurons), as well as introducing non-linearity.
- Hidden layer: a function $\mathbf{z}(x) : \mathbb{R}^d \to \mathbb{R}^J$ consisting of J neurons $\{z_j\}_{j=1}^J$ which all take the same input $x \in \mathbb{R}^d$. Every neuron has its own trainable weights and biases.

A feedforward neural network can be defined as:

Definition 3.2.1. Let $\{\mathbf{z}_l\}_{l=1}^L$ be L hidden layers such that $\mathbf{z}_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ for $l = 1, ..., L$, where N_l is the number of neurons for hidden layer l. Let the input $x \in \mathbb{R}^{N_0}$. The map $NN : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ given as

$$NN(x) = z_L(z_{L-1}(...z_1(x))) \tag{3.6}$$

is called a feedforward neural network.

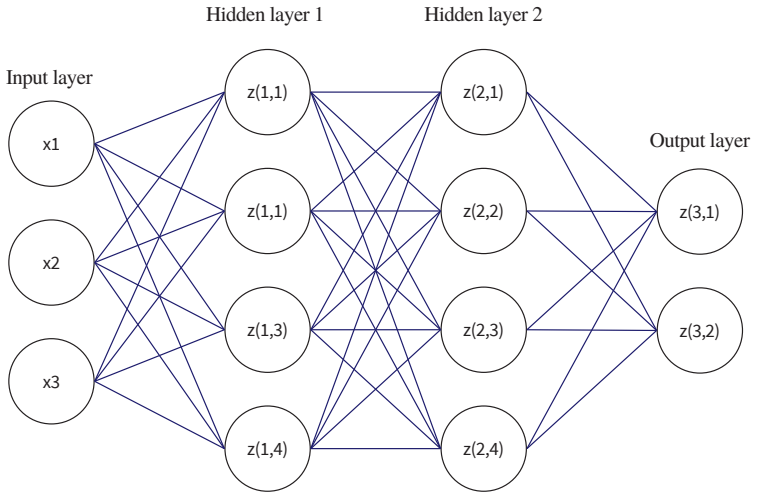


Figure 3.1: Illustration of feedforward neural network.

Figure 3.1 illustrates an example of how a neural network can be structured. Each node receives all outputs of the previous layer represented by the connecting lines, and the vertical stacks of nodes make up the hidden layers. The third hidden layer can be called the *output layer* since it is the final layer producing the output. The input is $x \in \mathbb{R}^3$, and the hidden layers are the following functions: $z_1 : \mathbb{R}^3 \to \mathbb{R}^4$, $z_2 : \mathbb{R}^4 \to \mathbb{R}^4$, $z_3 : \mathbb{R}^4 \to \mathbb{R}^2$.

The choice of activation function can affect both the training and performance of a neural network. As a rule, the same activation function is picked for every neuron in the network. As mentioned, activation functions must be non-linear and differentiable. Some common choices for such functions are:

- $\sigma(x) = \frac{1}{1+e^{-x}}$

- $\text{ReLU}(x) = \begin{cases} \text{x} & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$
- $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$

As mentioned above, the goal when training is to decrease training loss given training data $\{(x_i, y_i)\}_{i=1}^n$. The loss function is defined as

$$\text{Loss}(\theta|x, y) = \sum_{i=1}^{n} \|NN_\theta(x_i) - y_i\|_2^2. \tag{3.7}$$

Here, θ are the trainable parameters in the neural network, i.e. the weights and biases. In this case, the loss is squared l_2-distance, but other loss functions could have been chosen. The training of a neural network is commonly done through the *stochastic gradient descent algorithm*. The goal of this algorithm is to identify the minimum value of the loss function, or more precisely to find the weights and biases θ that give the lowest training loss $\text{Loss}(NN_\theta|x, y)$. Another way of putting it is it finds the neural network which produces output values that are as close as possible to the real output values. A simple version of the algorithm is given:

Algorithm 1 Stochastic gradient descent algorithm (SGD) [GBC16] Algorithm 8.1

- Input values: Training data $\{(x_i, y_i)\}_{i=1}^n$, number of epochs E, batch size b, learning rate or step size η

for epoch $= \{1, ..., E\}$ **do**

- Create a new permutation $\{p_1, ..., p_n\}$ of $\{1, ..., n\}$

for $i = (1, ..., \lceil n/b \rceil)$ **do**

- Select b pairs from the shuffled dataset $X_{\text{batch}} = \{(x_{p_j}, y_{p_j})\}_{j=ib}^{(i+1)b}$
- Compute gradient of the loss with respect to θ: $G = \nabla \text{Loss}(NN_\theta|X_{\text{batch}})$
- Update parameters $\theta \leftarrow \theta - \eta G$

end for

end for

Given certain assumptions for the neural network, the SGD algorithm converges to zero loss on the training data [Du+19]. The step where the gradients are computed with respect to the parameters θ, i.e. the weights and biases, is done using *backpropagation*. Essentially, backpropagation uses the chain rule to compute the gradient of the loss function with respect to every weight and bias. There exist more advanced versions of the SGD algorithms which use the momentum of the previous iterative steps in deciding the step size. The *Adam* optimizer [KB14], a name derived from *adaptive moment estimation*, updates the step size in the stochastic gradient descent algorithm every iteration instead of using a fixed step size as described in Algorithm 3.2. The learning rate is cleverly updated based on gradients in previous steps, leading to a more efficient search for the minimum of the loss function. The Adam optimizer is used later in this thesis.

One of the advantages of neural networks is their ability to approximate any continuous function well. It has long been known that a neural network with a single hidden layer is able to approximate any continuous function arbitrarily well [Hor91]. It has also been shown that multilayer feedforward networks with a non-polynomial activation function can approximate any function [Les+93]. Neural networks have been proven as an effective model for analyzing dynamical systems [Eid+23; GDY19].

3.3 Physics-informed machine learning

If we want to train a machine-learning model to learn a physical system from data, we may achieve a higher performance by letting the model assume the laws of physics before training, as opposed to training the model without any prior knowledge. This idea is called physics-informed machine learning and has in recent times been used to learn complex physical systems such as partial differential equations [Kar+21; RPK19]. A setback of complex machine learning methods such as deep neural networks (Section 3.2) is the large amount of data required to train the models. Because of the high cost of data acquisition for physical systems, it is of interest to learn such systems with a small amount of data available. Learning non-linear functions is especially challenging when the dataset is small and high-dimensional. Luckily, we often have prior knowledge of well-studied systems in physics and biology which we can use. By letting the model assume certain physical properties, the learning of a complex system can be simplified.

An example of such a physical assumption which is further explored later in this thesis is the conservation of energy in an isolated system. Assume we want to learn an ODE $\dot{x} = f(x)$ representing an isolated physical system from restricted data. If the machine-learning model $\hat{f}$ does not assume the law of energy conservation, it may make future predictions that exhibit a decreasing or increasing trend in energy over time. If we instead assume a Hamiltonian structure $\dot{x} = S(x)\nabla H(x)$ (see Section 2), we are guaranteed energy conservation as $\dot{H} = 0$. We can either learn both S and H as $\hat{\dot{x}} = \hat{S}(x)\nabla\hat{H}(x)$, or we could just approximate one function if we know the other. This is an example of how we leverage physical knowledge in learning. By enforcing the Hamiltonian structure in the learning, we have excluded all solutions which do not conserve energy, all of which are incorrect. This sparsity in the search space may help the machine-learning model arrive at the correct solution more quickly, especially on small datasets.

CHAPTER 4

System identification

In the field of system identification, the main idea is to extract patterns and knowledge about a mathematical system from data generated by the system. The topic has an extensive history and literature [Lju87] and has focused both on analyzing known systems and attempting to discover previously unknown systems from data. In this thesis, system identification means discovering the governing equations of an ODE, i.e. identifying the governing equations of dynamical systems of the form $\dot{x} = g(x)$. Given data on x and $\dot{x}$, the goal is to train a model $\hat{g}$ that not only makes accurate predictions but identifies the terms of g. Several breakthroughs have been made in applying machine learning techniques to system identification. Symbolic regression has been used [BL07; SL09], and in more recent times, a new approach called *sparse regression of nonlinear dynamics* (SINDy) has become popular [BPK16]. In this chapter, we go through these approaches as well as a gradient descent approach is the basis for the PHSI model.

4.1 Symbolic regression

Symbolic regression is a form of system identification that randomly combines mathematical building blocks such as analytical functions, operators, and constants to best fit the true solution to an ODE. Finding the terms that best fit g is a combinatorial problem, and search algorithms inspired by combinatorial optimization are used to search for the correct terms. [BL07; SL09] employ a version of the genetic algorithm. Algorithm 2 describes how the genetic algorithm can be applied to symbolic regression.

Algorithm 2 Genetic algorithm for symbolic regression

Input: Population size N, number of iterations MAX, measured data $\{x_j\}_{j=1}^m$ and $\{\dot{x}_j\}_{j=1}^m$

- Generate an initial population of candidate functions $\hat{g}_i,\ i = \{1, ..., N\}$
- Compute losses $L_i = \sum_{j=1}^m \|\dot{x}_j - \hat{g}_i(x_j)\|_2,\ i = 1, ..., N$
- Set $t = 1$

while $t <$ MAX **do**

- Select a pair from the population with probability related to loss
- Perform crossover operation on selected pair
- Perform mutation on offspring with mutation probability
- Replace the previous generation with the newly generated population
- Compute losses $L_i = \sum_{j=1}^m \|\dot{x}_j - \hat{g}_i(x_j)\|_2,\ i = 1, ..., N$
- $t = t + 1$

end while

- Return the candidate function with the lowest loss $\min_{i=1,...,N} L_i$

Each initial member of the initial population is a random combination of functions, constants, and operations from the functions space. The crossover operation is a random combination of the two parent functions, and the mutation is a random alteration of the terms of the offspring function [Koz94]. After the algorithm has been run, the function in the population that best fits the data is returned.

4.2 Sparse regression

Another method for discovering dynamical systems is sparse regression, as done by [BPK16] who name their method *Sparse Identification of Nonlinear Dynamical Systems* (SINDy). The idea is to search for the governing equations in a high-dimensional predetermined non-linear function dictionary with the assumption that the governing equations are sparse in the function dictionary. This is done as a regression problem as opposed to a combinatorial optimization problem. Although there is no limit to the size or scope of the function dictionary, in practical cases, the number of terms that describe an ODE is relatively few, meaning we can rely on a relatively small function dictionary. Given the ODE

$$\dot{x}(t) = g(x(t)) \tag{4.1}$$

where $x : [0, \infty) \to \mathbb{R}^n$ is a function of time $t > 0$, we measure data at time-points $t_1, ..., t_m$ to form the data-matrix

$$X = \begin{bmatrix} x_1(t_1) & \dots & x_n(t_1) \\ \vdots & \ddots & \vdots \\ x_1(t_m) & \dots & x_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}. \tag{4.2}$$

We also have data on the data differentiated with respect to time

$$\dot{X} = \begin{bmatrix} \dot{x}_1(t_1) & \dots & \dot{x}_n(t_1) \\ \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dots & \dot{x}_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}. \tag{4.3}$$

We then define a dictionary function $\Theta : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times p}$ taking a data matrix X as input and producing a dictionary of p non-linear functions. For example, $\Theta(X)$ can be constants, trigonometric terms, and polynomial terms:

$$\Theta(X) = \begin{bmatrix} | & | & | & | & & | & | \\ 1 & X & P_2(X) & P_3(X) & \dots & \sin(X) & \cos(X) \\ | & | & | & | & & | & | \end{bmatrix} \tag{4.4}$$

where higher polynomials are denoted as $P_2(X), P_3(X)...$ and include all cross-terms between elements of x. For instance,

$$P_2(X) = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & ... & x_n(t_1)^2 \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & ... & x_n(t_2)^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & ... & x_n(t_m)^2 \end{bmatrix},$$

and notation $\sin(X)$ is short for

$$\begin{bmatrix} \sin\left(x_1(t_1)\right) & \dots & \sin\left(x_n(t_1)\right) \\ \vdots & \ddots & \vdots \\ \sin\left(x_1(t_m)\right) & \dots & \sin\left(x_n(t_m)\right) \end{bmatrix} \tag{4.5}$$

and so forth. Each entry of $\Theta(X)$ represents a candidate function for the true g. Then, the coefficient matrix $\Xi = [\xi_1\ \xi_2\ ...\ \xi_n]$ determines which terms in the function dictionary are to be active to recreate g in Equation 4.1:

$$\dot{X} = \Theta(X)\Xi. \tag{4.6}$$

Note that in practical cases, $\dot{X}$ is not always available, and will have to be estimated numerically from X. The goal is to solve the set of parameters Ξ from data. We can split the problem of solving the matrix Ξ into solving each column $\xi_1, ..., \xi_n$ separately. Then, for $i = 1, ..., n$, our goal is to solve the vector ξ_i so that

$$\dot{x}_i = \Theta(X)\xi_i \tag{4.7}$$

where $\dot{x}_i = [x_i(t_1), ..., x_i(t_m)]^T$. In sparse regression, we want to utilize the prior knowledge that the true solution is sparse in the search space $\Theta(X)$, thus, we want most of the entries of Ξ to be zero. SINDy proposes two sparse regression algorithms: the *least absolute shrinkage and selection operator* (LASSO) [Tib96] and the *sequential thresholded least-squares algorithm* [BPK16].

Lasso

LASSO is an l_1-regularized regression that minimizes the square error plus a penalty term:

$$\xi_i = \underset{\xi}{\operatorname{argmin}} \|\dot{x}_i - \Theta(X)\xi\|_2^2 + \lambda\|\xi\|_1 \tag{4.8}$$

where $\lambda > 0$ is the weight of the penalty. This l_1-regularization promotes sparsity in the solution as explored in Section 4.4. There are no analytical solutions to solving this minimization problem, but a solution can be approximated in several ways, for instance through the *iteratively reweighted least squares algorithm* [Dau+10].

Sequentially thresholded least-squares

The sequential thresholded least-squares algorithm uses Ridge-regression iteratively. Instead of solving the minimization problem in Equation 4.8, Ridge regression solves

$$\xi_i = \underset{\xi}{\operatorname{argmin}} \|\dot{x}_i - \Theta(X)\xi\|_2^2 + \lambda\|\xi\|_2^2 \tag{4.9}$$

where the penalizing term has a l_2-norm instead of a l_1-norm. The Ridge regression problem has an analytical solution, which will be shown.

Theorem 4.2.1. *Assume we have the matrix $A \in \mathbb{R}^{m\times p}$, the vector $y \in \mathbb{R}^m$ and the constant $\lambda > 0$. Then, the following is true for the vector $\beta \in \mathbb{R}^p$:*

$$\underset{\beta}{\operatorname{argmin}} \|y - A\beta\|_2^2 + \lambda\|\beta\|_2^2 = (A^TA + \lambda I)^{-1}A^Ty \tag{4.10}$$

Proof: We start by rewriting the left-hand side expression to matrix form and calculating:

$$J(\beta) := \|y - A\beta\|_2^2 + \lambda\|\beta\|_2^2 = (A\beta - y)^T(A\beta - y) + \lambda\beta^T\beta \tag{4.11}$$
$$= (A\beta)^T(A\beta) - (A\beta)^Ty - y^T(A\beta) + y^Ty + \lambda\beta^T\beta \tag{4.12}$$
$$= \beta^TA^TA\beta - 2(A\beta)^Ty + y^Ty + \lambda\beta^T\beta \tag{4.13}$$

We perform matrix calculus to find the β that minimizes this expression. Keep in mind the following differentiation rules:

- Rule 1: for vectors a, b,

$$\frac{\partial a^Tb}{\partial a} = \frac{\partial b^Ta}{\partial a} = b$$

- Rule 2: for vector b and matrix B,

$$\frac{\partial b^TBb}{\partial b} = (B + B^T)b$$

We now differentiate J with respect to β using rule 1 and 2

$$\frac{\partial J}{\partial \beta} = \frac{\partial}{\partial \beta}\left[\beta^TA^TA\beta - 2(A\beta)^Ty + y^Ty + \lambda\beta^T\beta\right] = 2A^TA\beta - 2A^Ty + 2\lambda\beta \tag{4.14}$$

We compute the second derivative:

$$\frac{\partial^2 J}{\partial \beta^2} = \frac{\partial}{\partial \beta}\left[2A^TA\beta - 2A^Ty + 2\lambda\beta\right] = 2(A^TA + \lambda I) \tag{4.15}$$

Since $A^TA + \lambda I$ is positive definite (A^TA is positive semi-definite and λI is positive definite because $\lambda > 0$), any stationary point of J must be a minimum. We find this minimum:

$$\frac{\partial J}{\partial \beta} = 0 \tag{4.16}$$

$$2A^T A\beta - 2A^T y + 2\lambda\beta = 0 \tag{4.17}$$

$$(A^T A + \lambda I)\beta = A^T y \tag{4.18}$$

$$\beta = (A^T A + \lambda I)^{-1} A^T y \tag{4.19}$$

Since $A^T A + \lambda I$ is positive definite, it is invertible, meaning the analytical solution $(A^T A + \lambda I)^{-1} A^T y$ always exists. ■

If we apply Theorem 4.2.1 to the problem of Ridge regression (Equation 4.9) by letting $y = \dot{x}_i$ and $A = \Theta(X)$, we arrive at the analytical solution for $\Xi = [\xi_1\ \xi_2\ ...\ \xi_n]$ in Equation 4.6:

$$\xi_i = \left(\Theta(X)^T \Theta(X) + \lambda I\right)^{-1} \Theta^T \dot{x}_i, \quad i = 1, ..., n, \tag{4.20}$$

We are now ready to describe the sequentially thresholded least-squares algorithm which solves Ridge regression iteratively.

Algorithm 3 Sequentially thresholded least-squares algorithm [BPK16]

- Input values: regularization constant λ, data $X = [x_1, ..., x_n]$ and $\dot{X} = [\dot{x}_1, ..., \dot{x}_n]$, initial function dictionaries $[\Theta_1^0, ..., \Theta_n^0]$, number of runs MAX
- Find initial guesses ξ_i^0 by inserting $\dot{x}_i$ and Θ_i^0 into Equation 4.20 for $i = 1, ..., n$

for $k = (1, ..., \text{MAX})$ **do**
  for $i = (1, ..., n)$ **do**
    - Find indices whose absolute value in ξ_i^{k-1} is less than λ
    - Update the function dictionary Θ_i^k by removing these indices from Θ_i^{k-1}
    - Compute ξ_i^k by inserting $\dot{x}_i$ and Θ_i^k into Equation 4.20
  end for
end for

- Return $\Xi^{\text{MAX}} = [\xi_1^{\text{MAX}}, ..., \xi_n^{\text{MAX}}]$

The sequential thresholded least-squares algorithm (Algorithm 3) begins by finding the solution to $\Xi^0 = [\xi_1^0, ..., \xi_n^0]$ using Equation 4.20, and then thresholds all coefficients smaller than the cutoff value λ. Then, we remove the cutoff indices from the function dictionary before we find the least squares solution Ξ_1 using the updated function dictionary, and so on. The algorithm causes sparsity by repeatedly removing candidate functions from the search dictionary. This is a computationally inexpensive algorithm that rapidly converges to a sparse solution [BPK16].

4.3 Gradient descent

A third way of performing system identification is through stochastic gradient descent, i.e. the same algorithm used for training neural networks (see Algorithm 3.2 in Section 3.2). Again, we are trying to approximate an ODE

$$\dot{x} = g(x), \quad x \in \mathbb{R}^n \tag{4.21}$$

from time-series data. The goal of the gradient descent is to minimize the difference between g and a system identification model $\hat{g}$. Given data on x, we would ideally like to do this by measuring the loss between $g(x)$ and $\hat{g(x)}$. However, data on $g(x)$ (i.e. data on $\dot{x}$) is not available in practical cases, and we do not assume it to be known here. Instead, we define a loss function based on numerical integration. This requires some basic theory.

Numerical integration for solving ODEs

Consider an ODE $\dot{x} = g(x)$. Given $x(a)$ measured at time point $a > 0$, we can estimate the $x(b)$ where $b > a \geq 0$ using the integration scheme

$$x_b = x(a) + (b - a)\Psi(g, x(a), x_b, a, b) \tag{4.22}$$

where $x_b \approx x(b)$ and Ψ is a numerical integrator. This equation is equivalent to solving

$$\frac{x_b - x(a)}{b - a} = \Psi(g, x(a), x_b, a, b) \tag{4.23}$$

for x_b. If Ψ does not depend on x_b and can be directly computed from $x(a), a$, and b, we call it an *explicit* integrator. If, on the other hand, Ψ depends on x_b, we say it is an *implicit* integrator. In this case, x_b has to be solved from either an equation or a system of equations. A simple example of an explicit integrator is the forward Euler method:

$$\Psi(g, x(a), x_b, a, b) = g(x(a), a), \tag{4.24}$$

and a simple example of an implicit method is the backward Euler method:

$$\Psi(g, x(a), x_b, a, b) = g(x_b, b), \tag{4.25}$$

where x_b has to be solved. When using implicit integrators for solving x_b, we often use root-finding algorithms like Newton's method. We say that Euler's methods are of *order* 1. In general, an arbitrary numerical integrator $\Psi(g, x(a), x(b), a, b)$ is of order p if

$$x(b) - [x(a) + (h)\Psi(g, x(a), x(b), a, b)] = \mathcal{O}(h^{p+1}).$$

for any $b > a \geq 0$ where h is defined as $b - a$.

Now, consider that we have both the point $x(a)$ and $x(b)$ available and we have a function $\hat{g}$ approximating g. Then, given a numerical integrator Ψ, the right-hand side of the following equation should approximate the left-hand side

$$\frac{x(b) - x(a)}{b - a} \approx \Psi(\hat{g}, x(a), x(b), a, b) \tag{4.26}$$

Note that since $x(b)$ is known, we can compute $\Psi(\hat{g}, x(a), x(b), a, b)$ explicitly for a large subclass of implicit integrators, but not those that depend on intermediate steps that will have to be found implicitly. Equation 4.26 gives motivation for a loss function for a learning model $\hat{g}$: the better $\hat{g}$ approximates g, the better the right-hand side of Equation 4.26 approximates the left-hand side.

Loss function

We now define a loss function for the gradient descent. Assume that we have the data $X = [[x(a_1), x(b_1)], ..., [x(a_m), x(b_m)]]$ measured at m pairs of time points $\{a_j, b_j\}_{j=1}^m$ where $b_j > a_j \geq 0,\ j = 1, ..., m$. Given the system identification model $\hat{g}$, the data X and a numerical integrator Ψ, the following loss function is proposed:

$$\text{Loss}(\hat{g}_H; X, \Psi) = \sum_{j=1}^{m} \left\| \frac{x(b_j) - x(a_j)}{b_j - a_j} - \Psi(\hat{g}_H, x(a_j), x(b_j)) \right\|_2^2. \tag{4.27}$$

The motivation for the loss function is Equation 4.26 from the previous section. Similar integration schemes have been done for training dynamical system models in [Eid+23; Jin+20; MIY20].

System identification setup

Thanks to the gradient descent algorithm and backpropagation, we can impose any structure we would like on $\hat{g}$. Thus, we are not bound by the linear regression structure of SINDy, i.e. models that have the form of Equation 4.6. If we do not impose a specific structure on $\hat{g}$, we learn the terms of g directly, similarly to the SINDy approach (Section 4.2). We then define a function dictionary $\Theta_g : \mathbb{R}^n \to \mathbb{R}^p$ that takes the system coordinates $x \in \mathbb{R}^n$ as input, and outputs p non-linear candidate functions for g, which gives the learning model

$$\hat{g}(x) = (\Theta_g(x)\Xi_g)^T, \tag{4.28}$$

where the parameters $\Xi_g \in \mathbb{R}^{p \times n}$ are trained so that $\hat{g}$ best fits g.

Alternatively, we can impose a chosen structure on $\hat{g}$ based on our prior physical beliefs about the system. E.g. we can assume that the system is Hamiltonian (see Section 2), giving the ODE

$$\dot{x} = S(x)\nabla H(x). \tag{4.29}$$

If the structure matrix S is known, the system identification problem becomes learning the governing equations of the Hamiltonian, H. We denote this model as

$$\hat{g}_H(x) = S\nabla\hat{H}(x), \tag{4.30}$$

where $\hat{H}$ is the system identification model

$$\hat{H}(x) = \Theta_H(x)\Xi_H, \tag{4.31}$$

$\Theta_H : \mathbb{R}^n \to \mathbb{R}^p$ being p candidate functions for H, and $\Xi_H \in \mathbb{R}^p$ being the coefficients that are to be trained so that $\hat{H}$ best fits H. For instance, Θ_H can contain trigonometric terms and polynomial terms:

$$\Theta_H(x) = \begin{bmatrix} x & P_2(x) & P_3(x) & \dots & \sin(x) & \cos(x) \end{bmatrix} \tag{4.32}$$

where for example

$$P_2(x) = \begin{bmatrix} x_1^2 & x_1 x_2 & \dots & x_n^2 \end{bmatrix} \tag{4.33}$$

and

$$\sin(x) = \begin{bmatrix} \sin(x_1) & \dots & \sin(x_n) \end{bmatrix}. \tag{4.34}$$

The goal is to train the model coefficients Ξ_H through gradient descent so that $\hat{H}$ best represents H, or rather that $\nabla\hat{H}$ best represents ∇H. Though we do not give more examples here, any structure could be imposed on $\hat{g}$.

Training

We are then ready to train our model using stochastic gradient descent. As an illustration, we describe this algorithm for training a model with a Hamiltonian structure. However, the algorithm can be applied to any chosen structure of the model. Given the model $\hat{g}_H$, the stochastic gradient descent is given in Algorithm 4. The structure and training of $\hat{g}_H$ are actually similar to that of a single-layer feed-forward neural network (see Section 3.2 and Algorithm 3.2). But, it is important to note that neural networks are in general not system identification models. To avoid confusion, we will not call this system identification approach a neural network.

Algorithm 4 Learning a Hamiltonian system using stochastic gradient descent

- Input variables: input data $X = [[x(a_1), x(b_1)], ..., [x(a_m), x(b_m)]]$, number of epochs E, batch size b, integrator Ψ, initial model parameters Ξ_H, learning rate η

for e $= (1, ..., E)$ **do**

- Create new permutation $p = \{p_1, ..., p_m\}$ of $\{1, ..., m\}$

for $i = (1, ..., \lceil m/b \rceil)$ **do**

- Select b pairs from shuffled dataset $X_{\text{batch}} = \{[x(a_{p_j}), x(b_{p_j})]\}_{j=ib}^{(i+1)b}$
- Compute the gradient of the loss with respect to the model parameters $G = \frac{d\text{Loss}(\hat{g}_{\Xi_H}; X_{\text{batch}}, \Psi)}{d\Xi_H}$ through back-propagation
- Update model parameters $\Xi_H \leftarrow \Xi_H - \eta G_{\Xi_H}$

end for

end for

The clear advantage of this system identification model is the freedom of imposing any structure based on physical knowledge and prior beliefs. The approach was used by [DXZ20] in their method called *Sparse Symplectically Integrated Neural Networks* (SSINN). In Chapter 5, we define the PHSI model based on this approach.

4.4 Regularization and pruning

As discussed in Section 4.2, the true governing equations of a dynamical system are generally sparse in a defined function dictionary. Thus, sparse solutions of a system identification model should be desirable. Two tools that are known to promote sparsity are l_1-regularization and pruning. We discuss these further.

l_1-regularization

Given a system identification model $\hat{g}$ where $\Xi = [\xi_1, ..., \xi_N]$ are the model parameters, l_p-regularization means penalizing the size of the model parameters with the l_p-norm, e.g. by adding $\lambda\|\Xi\|_p$ to the loss function for $\hat{g}$ where λ is the weight of the penalty. The LASSO and Ridge regression techniques introduced in Section 4.2 are examples of this where p is 1 and 2, respectively. In addition to helping learning models avoid overfitting, regularization can promote sparsity in the model parameters. Perhaps the most intuitive form of sparse regularization is l_0-regularization where the penalty term is $\lambda \sum_{i=1}^{N} \mathbf{1}_{\theta_i \neq 0}$, i.e. counting the number of non-zero elements. However, minimizing this penalty is a combinatorial problem which usually is NP-hard [Nat95]. The most commonly used alternative is the l_1-regularization which is convex and can be solved easily. The penalty term becomes $\lambda \sum_{i=1}^{N} |\theta_i|$. In addition to being used in LASSO as mentioned above, it has been used in the field of compressed sensing where the goal is to acquire and reconstruct signals that are sparse in some domain [FR13]. It has also been used to successfully promote sparsity in neural networks [Ma+19], which gives a motivation to introduce l_1-regularization in the gradient-descent-based system identification model described in Section 4.3.

Pruning

Pruning algorithms have been used to promote sparsity in dynamical system models by [LTS22]. The idea of pruning a system identification model is to gradually shrink the function dictionary during training by iteratively eliminating candidate functions. The sequential thresholded least-squares approach [BPK16] can be considered a pruning algorithm. Given a system identification model $\hat{g}$, we sketch a simple magnitude-based algorithm in Algorithm 5 inspired by [LTS22].

Algorithm 5 Pruning algorithm [LTS22]

- Input: pruning interval P, pruning threshold ε, number of epochs E, system identification model $\hat{g}$ with model parameters Ξ

for e in $\{1, ..., E\}$ **do**
- Tune parameters of $\hat{g}$

  if $e//P = 0$ **then**
    for ξ in Ξ **do**
      if $|\xi| < \epsilon$ **then**
        - $\xi = 0$ for the rest of training

      end if
    end for
  end if
end for

The reason that the pruning only happens every P epochs of training is that we want the model parameters to converge between every pruning. In this way, we greatly reduce the risk of eliminating potentially correct candidate functions.

4.5 Advantages of the system identification approach

The system identification approach holds a distinct advantage over neural network techniques in terms of interpretability. While neural networks can achieve high accuracy in prediction given a sufficient amount of data and a sufficiently large network architecture [Cyb89; Hor91; Les+93], they do not provide a clear understanding of the underlying governing equations of the system being analyzed. In contrast, the system identification approach allows for discovering these governing equations, providing a deeper understanding of the system and better generalization. This can be especially beneficial when making predictions on new, unseen data as the model is able to leverage its understanding of the underlying equations to inform its predictions. This can lead to improved accuracy and more robust predictions compared to models that rely solely on pattern recognition. System identification models generally predict well on data that is sampled from outside of the domain of training data [BPK16].

PART II

Methodology and results

CHAPTER 5

Methodology

In this section, we briefly discuss how the theory of chapters 2 and 3 is combined to define the *pseudo-Hamiltonian system identification* model (PHSI). We discuss what assumptions will have to be made about a system before learning it using a PHSI model as well as the details of the training.

5.1 Pseudo-Hamiltonian System Identification

The core idea of our method is to learn pseudo-Hamiltonian systems using system identification. We introduce the pseudo-Hamiltonian system identification model (PHSI) which is a modification of the gradient descent-based system identification model described in Section 4.3. The goal of the PHSI model is to learn an ODE

$$\dot{x} = g(x,t), \quad x \in \mathbb{R}^n \tag{5.1}$$

where we assume that the ODE has a pseudo-Hamiltonian structure

$$\dot{x} = (S(x) - R(x))\nabla H(x) + F(x,t), \quad x \in \mathbb{R}^n.$$

The PHSI model learns every component of the system S, R, H, F separately and simultaneously from data. In this way, we do not only learn an approximation to the solution of the ODE, but we learn the structure of the system. We call $(S(x) - R(x))\nabla H(x)$ the *inner dynamics* of the system, while we may refer to the external forces $F(x,t)$ as the *outer dynamics*. We define the most general PHSI model as

$$\hat{g}_{\text{PHSI}}(x,t) := (\hat{S}(x) - \hat{R}(x))\nabla \hat{H}(x) + \hat{F}(x,t). \tag{5.2}$$

where $\hat{S}$, $\hat{R}$, $\hat{H}$ and $\hat{F}$ model the corresponding term in Equation (5.1). We want full insight into the inner dynamics of the system, thus $\hat{S}, \hat{R}$ and $\hat{H}$ are all learned through system identification models. Since we make no assumptions about the external forces, $\hat{F}$ will generally be a neural network due to its approximation ability. However, in cases where F is assumed to have a simple structure, it allows for learning it through system identification models as well. The main advantage of pseudo-Hamiltonian system identification is its ability to learn the inner dynamics of a system despite it being disturbed by damping and external forces. Since $\hat{H}$, $\hat{R}$, and $\hat{F}$ are learned separately, we can use $\hat{H}$ to study the behavior of the system without the presence of damping and external forces. For the numerical experiments in this thesis, we assume to know S

exactly, and $R(x)$ are assumed to be damping coefficients independent of x. Thus, Equation 5.2 is reduced to:

$$\hat{g}_{\text{PHSI}}(x,t) = (S - \hat{R})\,\nabla\hat{H}(x) \;+\; \hat{F}(x,t), \tag{5.3}$$

where

$$\hat{R} = \begin{pmatrix} \hat{r}_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \hat{r}_n \end{pmatrix} \tag{5.4}$$

is the matrix with the trainable elements of $\hat{R}$ on the diagonal. Since the learning model $\hat{g}_{\text{PHSI}}$ has a complex structure with several components to be trained, the gradient descent approach to system identification (see Section 4.3) is well suited due to its structural flexibility.

Learning Hamiltonian systems using system identification has been done before by [DXZ20] in their *Sparse Symplectically Integrated Neural Networks* (SSINN). The SSINN model, like the PHSI model, trains using gradient descent. However, SSINN assumes that the Hamiltonian is of canonical structure (see Equation 2.25) and that it is separable, meaning it can be split into kinetic and potential energy, (see Section 2.6). The PHSI model does not assume this separability and can learn Hamiltonian systems in the most general formulation (see Equation 2.26) in addition to learning damping and external forces. Thus, it allows for system identification on more general Hamiltonian systems than what has been done previously. [Eid+23] have also trained learning models of the pseudo-Hamiltonian structure, but they used multi-layer neural networks instead of system identification models.

System identification on the external forces and hybrid models

In the PHSI model, the approximation of the external forces $\hat{F}$ can be learned either by system identification or by a neural network. If the external forces are a complicated function with no simple analytical representation, they are best modeled by a neural network due to their approximation abilities, as discussed in Section 3.2. However, if the external forces have a simple analytic form, it is possible to learn these through system identification as well. In this case where $\hat{H}$ is modeled by system identification and $\hat{F}$ is modeled by a neural network, we call the PHSI model a *hybrid model*. One also has to assume whether $\hat{F}$ is state-dependent, time-dependent, or both. When $\hat{F}$ is a system identification model, the function dictionary can be chosen to be polynomial, trigonometric, or a combination of the two. The trigonometric function dictionary contains parameters for both the amplitude and period of the terms. For example, if we assume that the external port is strictly time-dependent and consists of a combination of trigonometric terms and a second-degree polynomial, the function dictionary is:

$$\hat{F}(t) = \xi_1 t + \xi_2 t^2 + \xi_3 sin(\xi_4 t) + \xi_5 cos(\xi_6 t) \tag{5.5}$$

where $\xi_1, ..., \xi_6$ are the trainable parameters of $\hat{F}$.

Regularization and pruning for promoting sparsity

Since we assume that the true governing equations are sparse in the search space, we employ the regularization and pruning techniques discussed in Section 4.4 and a pruning scheme based on Algorithm 5. The details of the implementation can be found in the training algorithm described in the next section. We also add l_1-regularization to $\hat{H}$. Since the Hamiltonian formulation is generally not unique (as discussed in Section 2.5), promoting sparsity in the learning of $\hat{H}$ favors the formulation that is most natural and interpretable.

Since $\hat{g}_{\text{PHSI}}$ learns the inner dynamics $\hat{H}$, the outer dynamics $\hat{F}$ and the damping effects $\hat{R}$ simultaneously, we apply regularization on $\hat{F}$ as well to guide the separation. As discussed in Section 2.7, there is generally not uniqueness in the separation of the pseudo-Hamiltonian system into internal and external dynamics. Thus, the PHSI model does not have prior knowledge about what part of the dynamics in the data should be attributed to H and F, respectively, unless assumptions about this are imposed. The goal is to learn the most natural representation of the system, where only the dynamics that do not fit $\hat{H}$ are attributed to the external forces. To achieve this, the regularization of $\hat{H}$ and $\hat{F}$ have to be well chosen as we discuss later in Section 5.2.

Training the PHSI model

In the PHSI model, we make the realistic assumption that we only have data on x, not $\dot{x}$. As explained in Section 4.3, we instead train on an integration scheme using a numerical integrator Ψ. Given time-series data $X = [[x(a_1), x(b_1)], ..., [x(a_m), x(b_m)]]$ measured at the pairs of time-points $\{(a_j, b_j)\}_{j=1}^m$, the loss is defined as

$$\begin{aligned}\text{Loss}(\hat{g}_{\text{PHSI}}; X, \Psi) = \sum_{j=1}^{m} \left\| \frac{x(b_j) - x(a_j)}{b_j - a_j} - \Psi(\hat{g}_{\text{PHSI}}, x(a_j), x(b_j)) \right\|_2^2 \\ +\lambda_H \|\Xi_H\|_1 + \lambda_F \|\Xi_F\|_1.\end{aligned} \tag{5.6}$$

where Ξ_H and Ξ_F are the model parameters for $\hat{H}$ and $\hat{F}$, respectively. We add l_1 regularization to $\hat{H}$ with weight λ_H and to $\hat{F}$ with weight λ_F. The choice of Ψ will be discussed below.

The training algorithm for PHSI is shown in Algorithm 6. It is similar to Algorithm 4 in that it is a stochastic gradient descent algorithm, but it is different in that it trains $\hat{H}, \hat{F}$ and $\hat{R}$ all at once.

Algorithm 6 Training algorithm for the PHSI model

- Input variables: input data $X = [[x(a_1), x(b_1)], ..., [x(a_m), x(b_m)]]$, number of epochs E, initial model parameters $\Xi_H, \Xi_F, \hat{R}$, learning rate η, integrator Ψ, batch size b, pruning interval P, pruning threshold ε

for e in $(1, ..., E)$ **do**
- Create a new permutation of $p = \{p_1, ..., p_m\}$ of $\{1, ..., m\}$

for i in $(1, ..., \lceil m/b \rceil)$ **do**
- Select b samples from the shuffled dataset $X_{\text{batch}} = \{[x(a_{p_j}), x(b_{p_j})]\}_{j=ib}^{(i+1)b}$
- Compute gradients of the loss with respect to the model parameters through backpropagation:
  - $G_{\Xi_H} = \frac{dL(\hat{g}_{\text{PHSI}}; X_{\text{batch}}, \Psi)}{d\Xi_H}$
  - $G_{\Xi_F} = \frac{dL(\hat{g}_{\text{PHSI}}; X_{\text{batch}}, \Psi)}{d\Xi_F}$
  - $G_{\hat{R}} = \frac{dL(\hat{g}_{\text{PHSI}}; X_{\text{batch}}, \Psi)}{d\hat{R}}$
- Update model parameters: $\Xi_H, \Xi_F, \hat{R} \leftarrow (\Xi_H, \Xi_F, \hat{R}) - \eta(G_{\Xi_H}, G_{\Xi_F}, G_{\hat{R}})$

end for

if $e//P = 0$ **then**

for ξ in $\{\Xi_H, \Xi_F, \hat{R}\}$ **do**

if $|\xi| < \varepsilon$ **then**
- $\xi = 0$ for the rest of the training

end if

end for

end if

end for

Initialization of trainable parameters

In Algorithm 6, the trainable parameters in the model, represented by Ξ_H, Ξ_F and $\hat{R}$ (where Ξ_H and Ξ_F are the model parameters of $\hat{H}$ and $\hat{F}$) are given initial values before they are tuned. The choice of how these parameters are initialized can have an effect on training. In the PHSI model, the parameters are initialized randomly. More precisely, all initial values of Ξ_H, Ξ_F, and $\hat{R}$ are i.i.d. from the Gaussian distribution $N(0, 0.5^2)$.

Choice of numerical integrator

As discussed in Section 4.3, a numerical integrator must be chosen when computing the loss for a PHSI model (see Equation 5.6). Given a PHSI model $\hat{g}$ and two data points $x(a), x(b)$ measured at time-points $b > a > 0$, the integrator is denoted as

$$\Psi(\hat{g}, x(a), x(b), a, b).$$

Perhaps the simplest example of such an integrator is the forward Euler integrator

$$\Psi(\hat{g}, x(a), x(b), a, b) = \hat{g}(x(a), a), \tag{5.7}$$

but this method only uses the observation $x(a)$. We instead seek a method that is dependent on and explicitly given by both $x(a)$ and $x(b)$ so that we use all the data available. Also, since we assume the dataset can be

contaminated with noise, we choose an integrator that is *symmetric*, i.e. $\Psi(\hat{g}, x(a), x(b), a, b) = \Psi(\hat{g}, x(b), x(a), b, a)$, meaning it depends equally on the two data points. An example of a symmetric integrator is the implicit midpoint Euler method:

$$\Psi(\hat{g}, x(a), x(b), a, b) = \hat{g}\left(\frac{x(a)+x(b)}{2}, a + \frac{b-a}{2}\right). \tag{5.8}$$

Since we assume both $x(a)$ and $x(b)$ have independent noise with standard deviation σ, the standard deviation of $(x(a)+x(b))/2$ in Equation 5.8 is $\sigma/\sqrt{2}$. In other words, we average out the noise from the two observations. Although the implicit midpoint method has the desired symmetry, it is only of order 2. Integrators of higher order tend to be more accurate but are generally computationally more expensive. An example of an integrator of order 4 is the *RK4*-method (also known as the classic Runge-Kutta method), which is given as

$$\Psi(\hat{g}, x(a), a, b) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5.9}$$

where

$$\begin{aligned} k_1 &= g(x(a), a) \\ k_2 &= g\left(x(a) + (b-a)\frac{k_1}{2}, \frac{a+b}{2}\right) \\ k_3 &= g\left(x(a) + (b-a)\frac{k_2}{2}, \frac{a+b}{2}\right) \\ k_4 &= g(x(a) + (b-a)k_3, b) \end{aligned} \tag{5.10}$$

However, the RK4 integrator does not have the symmetry that we seek. [Eid+23] propose an integrator that is both symmetric and of fourth order:

$$\begin{aligned} &\Psi(\hat{g}, x(a), x(b), a, b) = \\ &\frac{1}{2}\,\hat{g}\left(\frac{x(a)+x(b)}{2} - \frac{\sqrt{3}}{6}(b-a)\,\hat{g}(c_1 x(a) + c_2 x(b), a + c_2(b-a)),\ a + c_1(b-a)\right) \\ +&\frac{1}{2}\,\hat{g}\left(\frac{x(a)+x(b)}{2} + \frac{\sqrt{3}}{6}(b-a)\,\hat{g}(c_2 x(a) + c_1 x(b), a + c_1(b-a)),\ a + c_2(b-a)\right), \end{aligned} \tag{5.11}$$

for $c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}$ and $c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}$. This is the chosen integrator for the integration scheme in the PHSI model in all the numerical experiments.

5.2 A study of regularization in the PHSI model

In the case of learning dynamical systems of complex structures, regularization is important not only to promote sparsity but to ensure the desired structure of the ODE. We exemplify this by learning a mass-spring system using a PHSI model. A simple-mass spring system can be described through the following ODE:

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \frac{p}{m} \\ -kq \end{bmatrix} \tag{5.12}$$

Where we set weight $m = 1$ and spring stiffness $k = 1$. The system has the related Hamiltonian function

$$H(q,p) = \frac{1}{2}q^2 + \frac{1}{2}p^2 \tag{5.13}$$

which we can use to write the ODE as a Hamiltonian system with the canonical formulation

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial H}{\partial q} \\ \frac{\partial H}{\partial p} \end{bmatrix}. \tag{5.14}$$

The PHSI model assumes the following structure:

$$\hat{g}_{\text{PHSI}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \nabla \hat{H}(x) + \hat{F}(x). \tag{5.15}$$

As previously stated, the pseudo-Hamiltonian formulation is non-unique since a general ODE can be equivalently represented by different combinations of $\hat{H}$ and $\hat{F}$. Since system identification models attempt not only to accurately predict data but identify the governing equations, we are interested in separating $\hat{H}$ and $\hat{F}$ in a natural way. In this example, since there are no external forces present, the desired solution is $\hat{H}$ equals to Equation 5.13 and $\hat{F} = 0$.

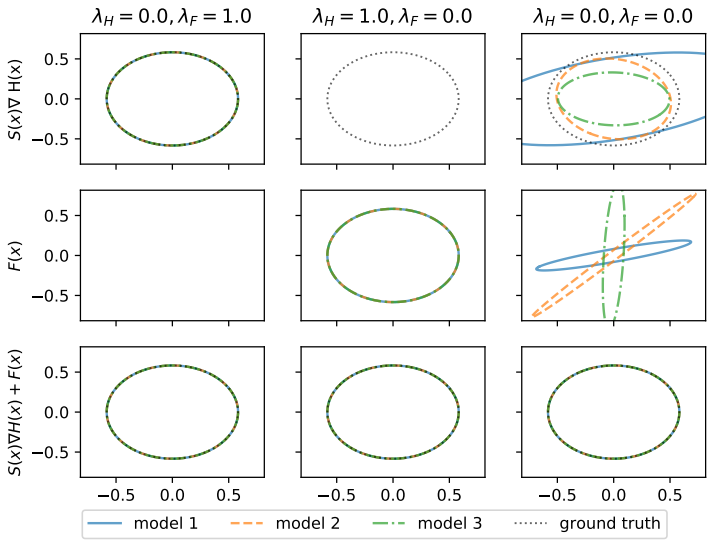


Figure 5.1: Phase plots of trajectories $x = (q, p)$ obtained from systems where $\hat{g}_{\text{PHSI}}$ is given by either $S(x)\nabla H$ or F or the sum of the terms. Each column is a different combination of regularization parameters, where λ_H and λ_F are the magnitudes of l_1 regularization on H and f, respectively. Three randomly initialized models are trained for each column.

We have trained system identification models with pseudo-Hamiltonian structures with different combinations of regularization on H and F. Figure 5.1 shows trajectory plots of three randomly initiated models compared to the

correct solution. Every model trained predicts $\dot{x}$ accurately, but only the models that regularize F are able to separate $\hat{H}$ and $\hat{F}$ in the desired way. The models that regularize H learn the entire ODE through F, and the models without regularization learn difficult-to-interpret combinations between H and F that are non-unique due to the random initialization of the model parameters. These results show how regularization affects the structure of a trained PHSI model. The way we set the regularization parameters of a PHSI model reflects the prior assumptions we make about the system to be learned.

5.3 Evaluation of the PHSI models

To assess the performance of the PHSI model, we compare its performance in numerical experiments to other dynamical system models discussed in Chapter 4. To be able to assess the advantage of the pseudo-Hamiltonian structure of PHSI, we also compare it against a baseline system identification model with no assumed structure.

Baseline system identification model (BSI)

We create a baseline system identification model (BSI) that has the same training strategy as the PHSI model but does not assume a pseudo-Hamiltonian structure. When comparing the performance of PHSI to BSI, we will be able to directly assess whether the PHSI model's pseudo-Hamiltonian structure has a positive, negative, or no effect on its performance. BSI employs the same pruning and regularization scheme as PHSI and trains on the same gradient-descent-based integration scheme described in Algorithm 6 and Equation 5.6. However, the baseline model does not assume a pseudo-Hamiltonian structure. Given the ODE

$$\dot{x} = g(x),$$

the BSI model simply tries to estimate g through a system identification model $\hat{g}$ with a predefined search space for g. It trains using the same numerical integrator as the PHSI model.

Comparison to other models

The PHSI model will be compared against three other system identification models: The SINDy model [BPK16] *Sparse Symplectically Integrated Neural Networks* (SSINN) of [DXZ20] introduced in Section 5.1, and finally the baseline gradient descent system identification model described above. In addition, we will make comparisons to the performance of the pseudo-Hamiltonian neural networks (PHNN) of [Eid+23]. The main attributes of these models are briefly summarized. When relevant, the public code for the models is provided.

- SINDy: Sparse Identification of Nonlinear Dynamical Systems. System identification method that learns g directly using sparse regression.[1]
- BSI: Baseline system identification. System identification model that learns g directly using a gradient descent algorithm.

[1] https://github.com/dynamicslab/pysindy

- SSINN: Sparse Symplectically Integrated Neural Networks: System identification model that trains using gradient descent. The model assumes a separable Hamiltonian (see Section 2.6), meaning it also implies that the Hamiltonian system has canonical structure (see Equation 2.25).[2]
- PHSI: pseudo-Hamiltonian system identification. System identification model that assumes g has a pseudo-Hamiltonian structure (see Equation 2.42). Trains using the gradient descent algorithm.
- PHNN: pseudo-Hamiltonian neural networks. Neural networks that assume the pseudo-Hamiltonian structure.[3]

[2] https://github.com/dandip/ssinn
[3] https://github.com/SINTEF/pseudo-hamiltonian-neural-networks

CHAPTER 6

Results

We now evaluate the PHSI model by training it on simulated data from four different numerical experiments. We also compare its performance to the four other models noted in Section 5.3. For each test, we train the relevant models on two datasets: one noise-free $X = \{x_i\}_{i=1}^N$ and one with Gaussian noise $X_{\text{noise}} = \{x_i + \varepsilon_i\}_{i=1}^N$ where $\varepsilon_i \sim N(0, \sigma^2),\ i = 1, ..., N$. We evaluate the models' predictive performance by measuring the l_2-error of simulated trajectories. These trajectories are not in the training data, hence can consider them to be validation data. The search space for the models consists of polynomial terms, trigonometric terms, or a combination of the two. The exact implementation details of each experiment can be found in Appendix B.

6.1 Experiment 1: Learning a Hamiltonian system

First, we test our methodology on data obtained from a pure Hamiltonian system, i.e. without damping or external forces. Thus, we can benchmark against existing methods while demonstrating the effect of the Hamiltonian framework. The experiments are done on the Hénon–Heiles system that was originally considered to describe the two-dimensional chaotic motion of stars around a galactic center [HH64]. It can be written as a Hamiltonian system of canonical form, and the Hamiltonian is separable, meaning it can be learned using the SSINN model. The Hamiltonian is defined as

$$H(q,p) = \frac{1}{2}(q_1^2 + q_2^2 + p_1^2 + p_2^2) + q_1^2 q_2 - \frac{1}{3}q_2^3, \tag{6.1}$$

and this can be used to form the canonical Hamiltonian system

$$\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{p}_1 \\ \dot{p}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \nabla H. \tag{6.2}$$

This is a separable system: $V(q) = \frac{1}{2}(q_1^2 + q_2^2) + q_1^2 q_2 - \frac{1}{3}q_2^3$, $T(p) = \frac{1}{2}(p_1^2 + p_2^2)$ and $H(q,p) = V(q) + T(p)$. To ensure a fair comparison, we train on the data generated and used in [DXZ20], although with more noise than was used in their experiments. The data consists of 3000 pairs of q, p at $t = 0$ and $t = 0.1$. The noisy data has $\sigma = 0.02$, which corresponds to approximately 3% of the maximum absolute values in the data. The system trajectory used for training

has initial conditions such that it is stable and does not diverge, as described in the paper [DXZ20]. The models were trained on a polynomial search space of third degree that includes all cross–terms between the four variables. In contrast to [DXZ20], the PHSI model does not assume a priori that the system is separable.

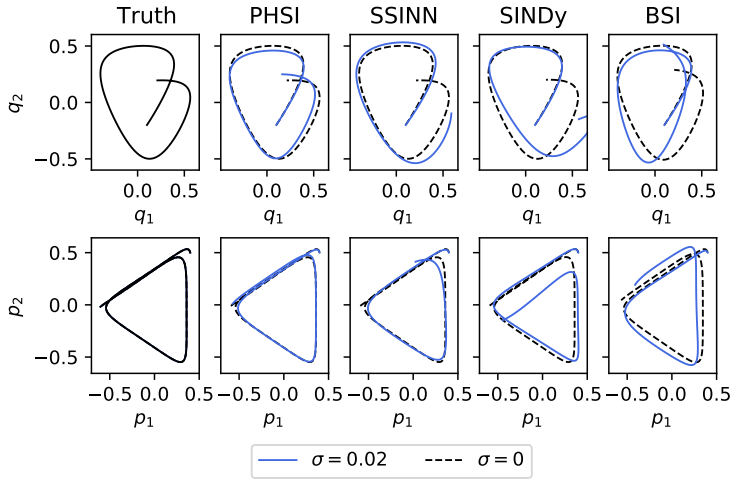


Figure 6.1: Comparison of simulated trajectories, with initial value $(q, p) = (0.1, -0.2, 0.4, 0.5)$, from $t = 0$ to $t = 10$.

Figure 6.1 shows example trajectories obtained from simulating the different learned systems in time. When training on noise-free data, all models learn the true coefficients up to 10^{-2}, while noise affects the different methods to different degrees. On the noisy data, PHSI outperforms SSINN by one order of magnitude when comparing the l_2-error of the trajectories, as depicted in Figure 6.2. PHSI and BSI both perform better than SSINN and SINDy, suggesting that the fourth-order symmetric integrator handles the noise well. The superior performance of PHSI over BSI suggests that imposing the Hamiltonian structure gives better performance yet. The SINDy and BSI models could not be converted into Hamiltonian functions when trained on noisy data. In addition, the solution of the SINDy model tends to become unstable over long time periods, something that PHSI and SSINN avoid by imposing a Hamiltonian structure.

Since the goal of system identification is not only accurate predictions but to learn the correct equations for the system, we have compared the learned coefficients of PHSI and SSINN in Table 6.1, where coefficients are learned from noisy data. It is clear that the PHSI model not only predicts more accurately as seen in Figure 6.2 but learned coefficients that are closer to representing the true Hamiltonian function.

Trained Parameters							
Model	q_x^2	q_y^2	$q_x^2 q_y$	q_y^3	p_x^2	p_y^2	$q_y p_x^2$
True Value	0.5	0.5	1	-0.333	0.5	0.5	0
PHSI	0.509	0.495	1.009	-0.338	0.501	0.477	0.124
SSINN	0.421	0.378	0.569	-0.195	0.484	0.443	N/A
	q_y	$q_x q_y$	p_y	$p_x p_y$			
True Value	0	0	0	0			
PHSI	0	0	0	0			
SSINN	0.004	0.002	-0.006	-0.006			

Table 6.1: Table comparing learned Hamiltonian coefficients for PHSI and SSINN to the exact values for the noisy data set with $\sigma = 0.02$.

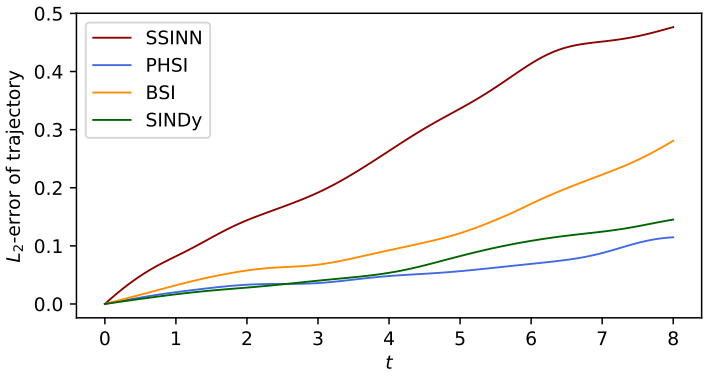


Figure 6.2: The average l_2-error of the trajectories obtained with 10 different random initial conditions, from the different models trained on noisy data with $\sigma = 0.02$, compared to trajectories simulated from the exact system.

6.2 Experiment 2: Learning a non-separable Hamiltonian system

A strength of the PHSI model is that it can learn systems with non-separable Hamiltonians. Previous methods for performing system identification on Hamiltonian systems (i.e. the SSINN model) assume separability of the Hamiltonian. The PHSI model, however, does not assume this and can learn the most general Hamiltonians. We will demonstrate this by testing its ability to learn a finite-dimensional nonlinear Schrödinger system, as considered in [Tao16]. This is a canonical system and has the Hamiltonian

$$\begin{aligned} H(q,p) =& \frac{1}{4}\sum_{i=1}^{d}(q_i^2+p_i^2)^2 \\ & -\sum_{i=2}^{d}(p_{i-2}^2 p_i^2 + q_{i-2}^2 q_i^2 - q_{i-1}^2 p_i^2 - p_{i-1}^2 q_i^2 + 4q_{i-1}q_i p_{i-1} p_i) \end{aligned}$$

where d is the number of particles in the system. We set $d = 2$, and the Hamiltonian becomes

$$\begin{aligned} H(q,p) =& \frac{1}{4}(q_1^2 + p_1^2)^2 \\ &+ \frac{1}{4}(q_2^2 + p_2^2)^2 - q_1^2 q_2^2 - p_1^2 p_2^2 + q_1^2 p_2^2 + q_2^2 p_1^2 - 4 q_1 q_2 p_1 p_2. \end{aligned} \tag{6.3}$$

The training set consists of 30 trajectories simulated for 100 time steps with step size 0.01. After being trained on the training set, the PHSI, SINDy, and BSI models, will be compared on their performance. Since the SSINN model can only learn separable Hamiltonians, it is not applicable here.

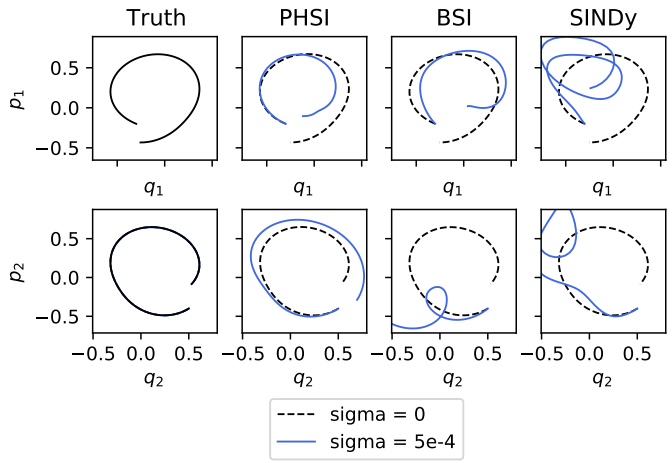


Figure 6.3: Phase portraits showing the trained models' trajectories next to the ground truth trajectory. All models are trained on the two training sets, one clean data set and one noisy with $\sigma = 10^{-4}$. The initial values are $(q, p) = (-0.3, 0.5, -0.2, -0.4)$.

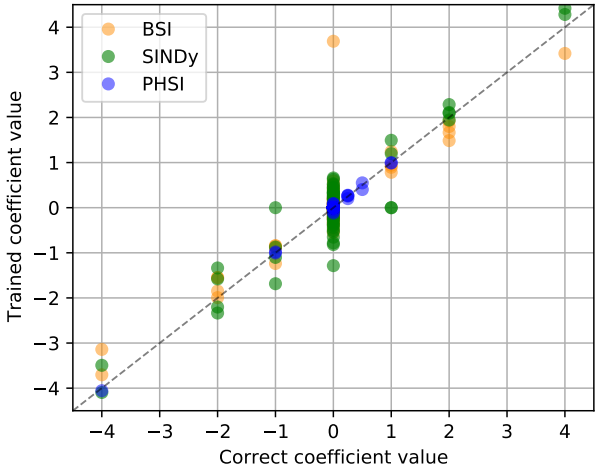


Figure 6.4: The trained coefficient values of PHSI, BSI, and SINDy plotted against the respective true values. A perfectly trained model only has points along the dotted line $y = x$. The models are trained on the noisy data set. Note that PHSI learns the Hamiltonian function while SINDy and BSI learn the right-hand side g of (1.1) and hence they do not learn the same coefficients for corresponding terms.

The PHSI and SINDy models were able to learn the true equations of the system on the noise-free data up to a precision of 10^{-5}, while the BSI model was able to learn to a precision of 10^{-3}. On the noisy data, the PHSI model was able to learn the true equations up to a precision of 10^{-1}. SINDy only learned the terms up to a precision of 1, and the BSI model achieved an even lower precision. Figure 6.3 compares the predictive abilities of the trained models. While all the models trained on noiseless data are able to predict trajectories accurately, the models trained on noisy data show different results. The PHSI model is not only more accurate, but it is also more stable. While the BSI and SINDy models seem to diverge quickly from the true trajectory, the predicted PHSI trajectory illustrates the energy-conserving properties of the Hamiltonian structure. Figure 6.4 confirms a more accurate representation of the true system equations in the PHSI model than SINDy when trained on noisy data. Neither the coefficients learned by SINDy nor BSI could be converted to a separable Hamiltonian, meaning the Hamiltonian is only preserved in the PHSI model due to its structure.

6.3 Experiment 3: Learning a pseudo-Hamiltonian system

We introduce a system with a pseudo-Hamiltonian structure. We consider a simple mass-spring system with damping and external forces. Given position $q \in \mathbb{R}$ and momentum $p \in \mathbb{R}$, the Hamiltonian of the mass-spring system is given by

$$H(q,p) = \frac{1}{2}q^2 + \frac{1}{2m}p^2, \tag{6.4}$$

where k and m are the stiffness constant and the mass of the spring, respectively. When adding damping and external forces, the system can be expressed with a pseudo-Hamiltonian formulation as

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -c \end{bmatrix} \begin{bmatrix} kq \\ \frac{1}{m}p \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha \sin(\omega t) \end{bmatrix}. \tag{6.5}$$

The damping coefficient c and the external forces $\alpha \sin(\omega t)$ act directly on the momentum in the system. We let $k = m = 1$, $\alpha = 2$, $\omega = 0.5$ and $c = 0.3$. We simulate data from this system and let the data consist of 50 trajectories from time 0 to 10 with time step 0.1. The noisy dataset has $\sigma = 0.2$. We trained three system identification models on these datasets: PHSI, BSI, and SINDy. In addition, we trained a PHNN model. PHSI models the Hamiltonian with $\hat{H}$ and the external forces with $\hat{F}$ (see Equation 5.2). The function dictionary for $\hat{H}$ is third-degree polynomials and the function dictionary for $\hat{F}$ is third-degree polynomials plus trigonometric functions. $\hat{F}$ is assumed to be strictly time-dependent.

Note that the SINDy model can only learn the amplitude of the external force, not the frequency. This is due to the regression structure of the SINDy model (see Chapter 4) which only allows for learning the coefficients multiplied with each function in the function dictionary. Thus, we do not expect SINDy to perform well in this experiment unless the frequency is already known.

Trained PHSI parameters					
	q^2	p^2	c	α	ω
True Value	0.5	0.5	0.3	2	0.5
$\hat{H}$	0.473	0.484			
$\hat{R}$			0.300		
$\hat{F}$	0	0	0	1.984	0.505
Trained BSI and SINDy parameters					
	q	p	*const.*	α	ω
True value	0	1	0	0	0
BSI $\hat{\dot{q}}$	0	0.991	0	0	0
SINDy $\hat{\dot{q}}$	0	0.983	0	0	0
True value	-1	-0.3	0	2	0.5
BSI $\hat{\dot{p}}$	-0.980	-0.278	0	1.999	0.506
SINDy $\hat{\dot{p}}$	-0.548	0	0.241		

Table 6.2: Learned coefficients for the mass-spring problem on noiseless data. q and p are multiplied with the trained coefficients while c, α, ω and *const.* are themselves the trainable parameters. Empty entries mean that the model does not learn that term.

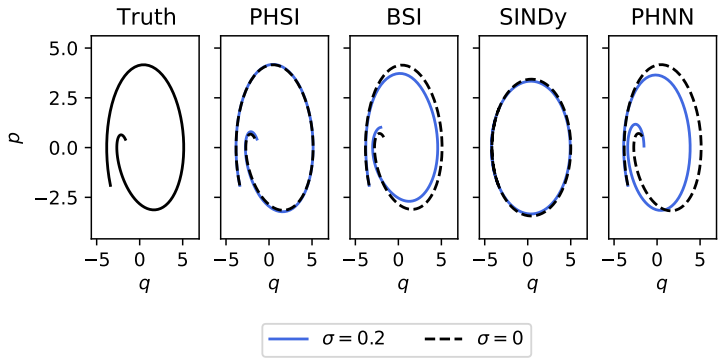


Figure 6.5: Comparison between the phase portraits obtained from integrating the exact system and the learned models from time 0 to 10. The initial value is $(q, p) = (-3.4, -1.9)$.

Figure 6.5 shows the predictive ability of each of the trained models. The PHSI, BSI, and PHNN models all perform well when trained on noiseless data. As expected, SINDy is not able to learn the effect of the external forces correctly and does not predict the trajectory with the same accuracy. When trained on noisy data, the PHSI model outperforms both the BSI and PHNN models.

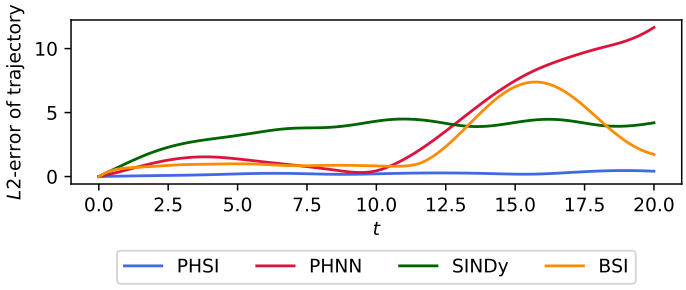


Figure 6.6: Average l_2-error of 30 simulated trajectories, with random initial conditions. These simulated trajectories have a longer time span than the training trajectories.

Figure 6.6 compares the average l_2 error of 30 different simulated trajectories of each of the models trained on noisy data. Since the trajectories are sampled from time 0 to 20, and the training trajectories are only sampled from time 0 to 10, this figure shows the models' predictive ability outside the training domain. As expected, the SINDy model predicts poorly. Now, let us compare the predictions of the PHSI and PHNN models. The PHSI model accurately predicts the true trajectories from time 0 to time 20 although it has only trained on trajectories from time 0 to 10. The PHNN model, however, only predicts well for the first 10 seconds of the trajectories. Then, from time 10 to 20, the predictions worsen sharply. This is an illustration of one of the advantages system identification models have over neural network models as discussed in Chapter 4. The neural network model approximates the patterns of the

true system well from time 0 to 10 but is not informed enough to predict the system's behavior from time 10 to 20 because it has not trained on this domain. The PHSI model, however, accurately learns the true equations of the system as shown in Table 6.3 which allows it to have informed predictions outside of the domain of training data. Although the BSI model is a system identification model, it also suffers from poor predictions between times 10 and 20, suggesting that the BSI model did not learn a good representation of the true system. This provides a qualitative argument in favor of using a model that assumes a pseudo-Hamiltonian formulation.

Note the PHSI model does not only achieve accurate predictions but is able to learn the true governing equations of both the inner dynamics, the damping, and the external forces. Thus, we gain full insight into the dynamics which again can enhance our physical understanding of the system. In addition, we can accurately simulate the behavior of the system where any of the components can be modified as we like. Alternatively, we can simulate the system without damping and external forces altogether.

6.4 Experiment 4: Hybrid model combining system identification with neural networks

In reality, many systems will be affected by external forces for which it can be difficult to find analytic terms. The final system to be studied consists of N tanks connected by M pipes, also considered by [Eid+23], with leaks that can be viewed as external forces. The damping effect in the pipes is assumed to depend linearly on the flow, and the total energy, i.e. the Hamiltonian is given by

$$H(\phi, \mu) = \sum_{i}^{M} \frac{1}{2J_i}\phi_i^2 + \sum_{j}^{N} \frac{g\rho}{2A_j}\mu_j^2, \tag{6.6}$$

where ϕ_i is the flow in pipe i scaled by a factor J_i depending on the density of the fluid and the dimension of the pipe, μ_j is the volume of the fluid in tank j, g is the gravitational constant, ρ is the density of the fluid, and A_j is the footprint of tank j. We thus have the pseudo-Hamiltonian formulation

$$\begin{bmatrix}\dot{\phi}\\ \dot{\mu}\end{bmatrix} = \begin{bmatrix}-\mathrm{diag}(r_p) & B^T\\ B & 0_{N\times N}\end{bmatrix}\begin{bmatrix}\frac{\partial H}{\partial \phi}\\ \frac{\partial H}{\partial \mu}\end{bmatrix} + \begin{bmatrix}0_M\\ f(\phi,\mu)\end{bmatrix}, \tag{6.7}$$

where the incidence matrix $B \in \mathbb{R}^{M\times N}$ describes how the pipes and tanks are connected and $r_p \in \mathbb{R}^M$ contains the damping coefficients relating to each pipe. When simulating the system to generate the training data, we set $\rho = 1$, $J_i = 0.02\ \forall i$, $A_j = 1\ \forall j$, $r_p = (0.03, 0.03, 0.09, 0.05, 0.05)$. A leak in the fourth tank is described by $f(\phi, \mu) = (0, 0, 0, -10\min(0.3, \max(\mu_4, 0.3)))^T$.

Since the external forces have a form that is difficult to find analytically, they are modeled by a neural network as opposed to a system identification model. We still aim to find the analytical expression of the internal dynamics through system identification. The model assumes that the external forces

only affect the fourth tank. The neural network modeling the external forces consists of three hidden layers, 100 nodes per layer, and the ReLU activation function. The PHSI model will be compared to the SINDy and BSI models. All models were trained on a function dictionary of polynomials up to the second degree. The training sets consist of 60 trajectories simulated between times 0 and 0.5 with time step 0.01. The noisy dataset has $\sigma = 0.03$.

Trained PHSI parameters							
	x_1^2	x_2^2	x_3^2	x_4^2	x_5^2	x_6^2	x_7^2
True Value	25	25	25	25	25	4.905	4.905
PHSI	24.94	24.98	24.98	24.95	24.97	4.930	4.960
	x_8^2	x_9^2	r_1	r_2	r_3	r_4	r_5
True Value	4.905	4.905	0.03	0.03	0.09	0.05	0.05
PHSI	4.890	4.930	0.031	0.029	0.086	0.051	0.041

Table 6.3: Learned coefficients for the tank problem on the noisy data. $x_1^2, \ldots, x_9^2$ are multiplied with the trained coefficients while $r_1, \ldots, r_5$ are themselves the trainable parameters.

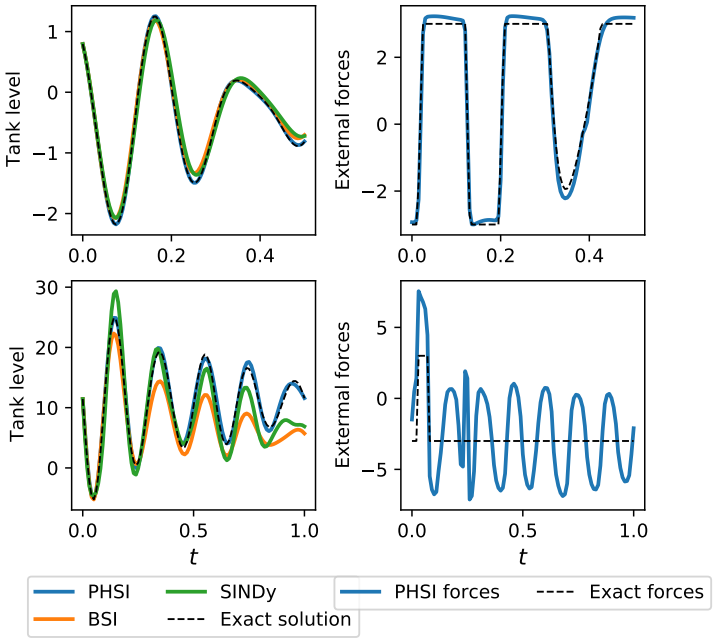


Figure 6.7: *Left:* The volume of the fluid in the leaky fourth tank simulated from the exact system and the different models. *Right:* The leak approximated by the neural network in the hybrid PHSI model, compared to the ground truth. The upper plots have initial conditions within the distribution of the training data: $(\phi, \mu) = (-0.4, 0, 0.5, 0, 0.2, 0 - 0.6, -0.5, 0.5)$. The lower plots show prediction on data outside the training data's time and space domain; time from 0 to 1 and initial state values $(\phi, \mu) = (10, 19, 4, 19, 7, 9, 17, 9, 11)$.

Figure 6.7 shows the predictive performance of the PHSI, SINDy, and BSI models. The upper plot shows that all the models predict the tank level well when the total tank volume is within the region of training data. The lower plots, however, show the predicted tank level on a system where the total tank volume is larger than the systems in the training set. This means we can evaluate the models' performance on data outside of the training domain. The accuracy of the BSI and SINDy models is greatly reduced, while the PHSI model is less affected. The reason for this can be found in Table 6.4: the PHSI model has accurately learned the true governing equations for the inner dynamics which helps inform predictions on the large-volume tank (see lower left plot of Fgure 6.7). The SINDy and BSI models have failed to learn the true governing equations with the same accuracy, and thus their predictions on the large-volume tank suffer. Figure 6.8 shows that the PHSI and BSI models clearly outperform the SINDy model in their predictive ability for the tank problem when trained on noisy data, suggesting that the gradient-descent based training combined with the integration scheme suggested by [Eid+23] handles the noise in the data better than the SINDy method. The fact that the PHSI model also outperforms the BSI model suggests that the proposed hybrid model outperforms a standard system identification approach.

This experiment shows the real strength of the PHSI model. When the inner dynamics of the ODE are disturbed by an external force, trying to learn the true governing equations of the inner dynamics is difficult. By modeling the external forces separately through a neural network, system identification of the inner dynamics becomes possible. We can then gain insight even in cases where the function describing the external forces is complex. In addition to allowing for interpretability, this approach results in more accurate predictions of the future states of the system, especially when the system is outside of the training data domain. We would also be able to accurately simulate the system's behavior without the presence of damping and external forces.

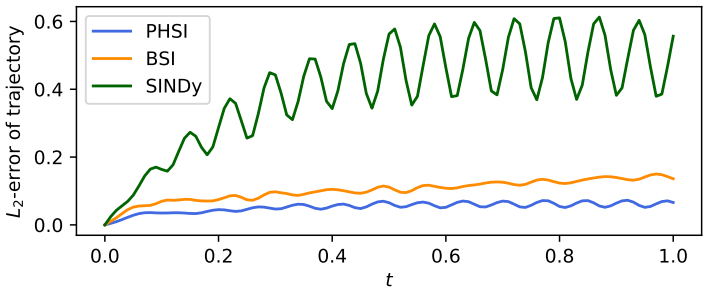


Figure 6.8: Average l_2-error of 30 simulated trajectories. The trajectories are sampled from the same distribution as the training set. The models are trained on the noisy dataset with $\sigma = 0.005$.

6.5 Study of regularization and pruning

In Chapter 3, it is argued that regularization and pruning help the PHSI model promote sparsity in search of the true governing equations of a dynamical system. It is also argued that since we know that the true governing equations are generally sparse in the function dictionary, promoting sparsity during training helps the PHSI model better approximate the true model for the governing equations. To evaluate whether this claim has merit or not, we train PHSI models where the amounts of regularization and pruning vary and the rest of the hyperparameters are held constant. By the amount of pruning, it is meant how often the pruning algorithm is employed during training, i.e. the value P in Algorithm 6. The lower value P has, the more frequent the pruning. By the amount of regularization, we specifically mean the amount of l_1-regularization on the Hamiltonian $\hat{H}$, i.e. λ_H in Equation 5.6. We evaluate the predictive performance of each trained model by comparing their *predictability score*, i.e. their average error over simulated trajectories with random initialization compared to the true trajectories, where the trajectories are not from the training set. The tests will be done for all of the four systems introduced in this chapter, and the PHSI model hyper-parameters can be found in Appendix B. The tests are performed on noiseless data.

Experiment 1: Hénon–Heiles system

The Hénon–Heiles system is a purely Hamiltonian system, and this is assumed when training the PHSI model. The models are trained on noiseless, simulated data with $\lambda_H \in \{0, 0.05, 0.5\}$ and $P \in \{1, 2, 4\}$, and the training amount is 3 epochs. The predictive performances of models with different combinations of regularization and pruning are shown in Figure 6.9. The figure shows that for this particular system, the PHIS model trains best without l_1-regularization, hence no regularization was used in the experiments in Section 6.1. The reason for this may lie in that the PHSI model learns the relatively simple Hénon–Heiles system in very few epochs of training. The pruning algorithm does not appear to affect the average l_2-error in this experiment, yet it excludes the correct terms resulting in easier interpretability of the trained model. Figure 6.10 shows that although the scores in Figure 6.9 differ by several orders of magnitude, all the models achieve a relatively good predictive ability.

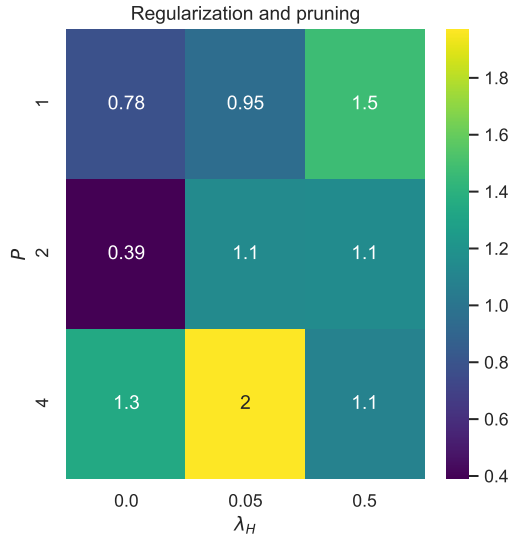


Figure 6.9: Average l_2-error over 30 simulated trajectories of the Hénon–Heiles system by PHSI models trained with different combinations of regularization-parameter λ_H and pruning-parameter P. The models are trained for 3 epochs, meaning when $P = 4$, the pruning never occurs.

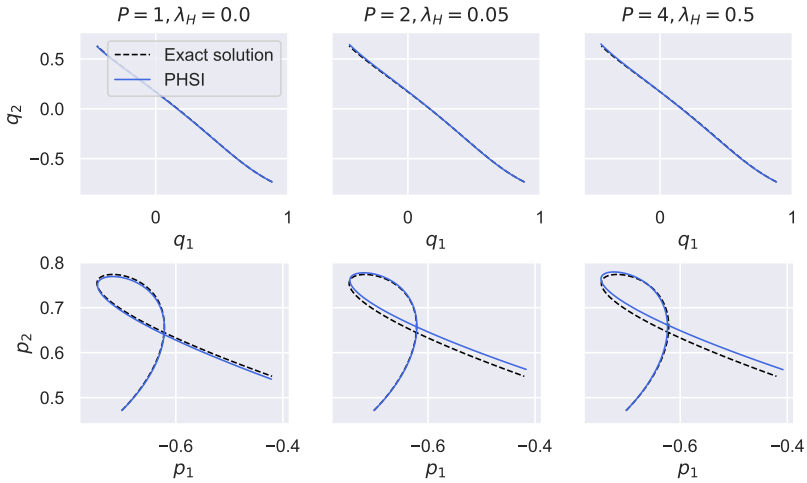


Figure 6.10: Simulated PHSI trajectories along with the ground truth of the nonlinear Hénon–Heiles system for three different combinations of regularization and pruning.

Experiment 2: Nonlinear Schrödinger system

The non-linear Schrödinger is also a purely Hamiltonian system, and this is assumed when training the PHSI model. The models are trained with $\lambda_H \in \{0, 0.05, 0.5\}$ and $P \in \{1, 5, 10\}$, and the they are trained for 10 epochs. In Figure 6.11, we observe that the regularization improves the model's performance drastically. The model with $P = 10$ and $\lambda = 0$ has an average l_2-error approximately 5 times that of the model with $P = 2$ and $\lambda = 0.5$, suggesting that the combination of regularization and pruning work well together. The reason that these results were different than for the Hénon–Heiles system may be that the non-linear Schrödinger system is non-separable and of a higher polynomial order, making the true solution more sparse in the function dictionary. Figure 6.12 illustrates the vastly superior performance of the models with more regularization and pruning over those with less.

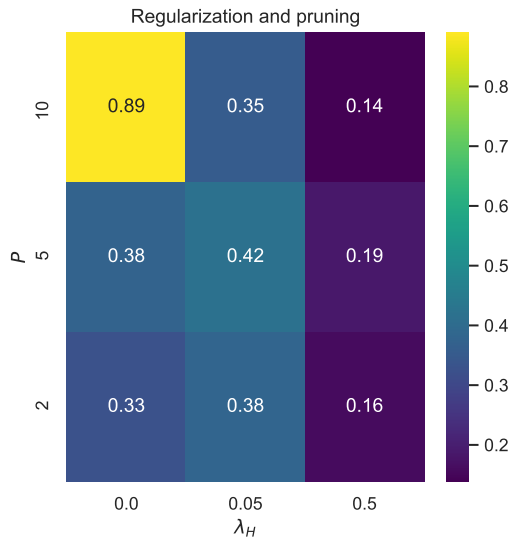


Figure 6.11: Average l_2-error over 30 simulated trajectories of the nonlinear Schrödinger system by PHSI models trained with different combinations of regularization-parameter λ_H and pruning-parameter P. The models are trained for 10 epochs, meaning when $P = 10$, the pruning only occurs once.

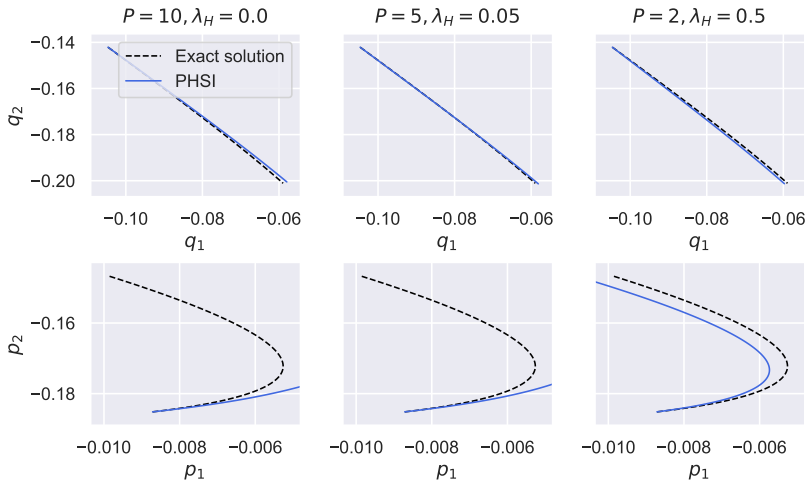


Figure 6.12: Simulated PHSI trajectories along with the ground truth of the nonlinear Schödinger system for three different combinations of regularization and pruning.

Experiment 3: Forced and Damped mass-spring system

The damped mass-spring system described in Section 6.3 is a pseudo-Hamiltonian system. That means we have regularization on the Hamiltonian and the external forces (λ_H, λ_F in Equation 5.6). Again, we choose to test the PHSI model for different values of λ_H and P with $\lambda_H \in \{0, 0.1, 0.5\}$ and $P \in \{20, 40, 80\}$, and the models will train for 80 epochs. λ_F is set to 0.01. Figure 6.13 indicates that promoting sparsity helps the PHSI model's predictive performance, especially through l_1-regularization. The superior predictive ability of the PHSI models that promote sparsity is also illustrated in the plotted simulated trajectories in Figure 6.14.

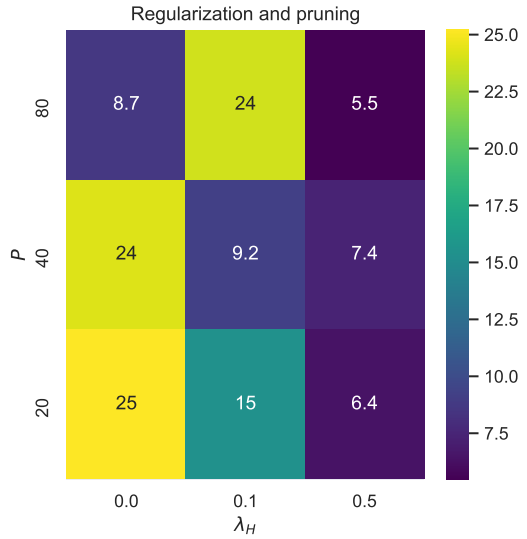


Figure 6.13: Average l_2-error over 30 simulated trajectories of the damped mass-spring system by PHSI models trained with different combinations of regularization-parameter λ_H and pruning-parameter P. The models are trained for 80 epochs, meaning when $P = 80$, the pruning only occurs once.

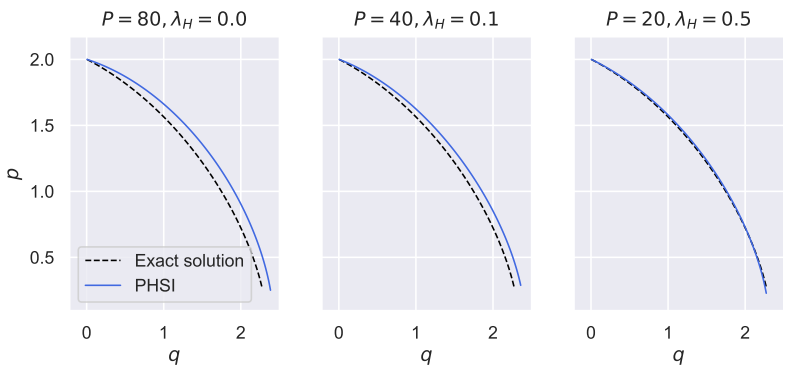


Figure 6.14: Simulated PHSI trajectories along with the ground truth of the damped mass-spring system for three different combinations of regularization and pruning.

Experiment 4: Tank system

The final system we study is the tank system from Section 6.4. We choose the search-spaces $\lambda_H \in \{0, 0.05, 0.5\}$ and $P \in \{10, 40, 80\}$, and the models are trained for 80 epochs. As shown in Figure 6.15, the regularization and pruning have a great impact on the performance of the model, and appear essential for achieving an accurate PHSI model. The model with $\lambda_H = 0$ and $P = 80$ has a predictive score about 60 times that of the model with $\lambda_H = 0.5$ and $P = 10$. Figure 6.15 confirms that this assumption is true for the tank system.

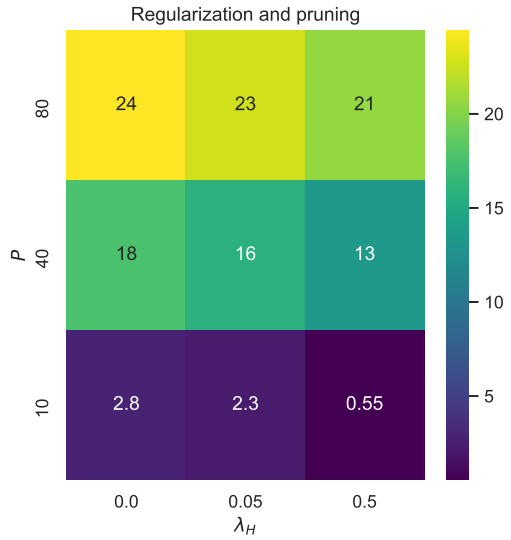


Figure 6.15: Average loss over simulated trajectories for PHSI models trained with different combinations of regularization-parameter λ_H and pruning-parameter P. The models are trained for 80 epochs, meaning when $P = 80$, the pruning only happens once.

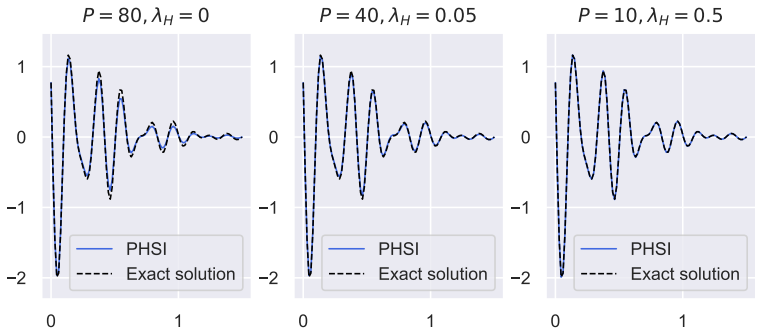


Figure 6.16: Simulated PHSI trajectories along with the ground truth of one of the leaking tanks in the connected tank system for three different combinations of regularization and pruning.

In conclusion, promoting sparsity in the function dictionary generally improves the accuracy of the PHSI model. Since promoting sparsity in the PHSI model through regularization and pruning improved results for three of the four models (it neither improved nor worsened the performance when learning the Hénon–Heiles), the sparsity assumption made in Chapter 3 seems to hold for a variety of systems. This again implies that the PHSI model is in general well fit for finding true governing equations. Note that the regularization and

pruning had more of an impact when learning systems of high dimensionality and polynomial order i.e. the nonlinear Schrödinger system and the tank system. If the PHSI model is to be applied to learning more complex and higher dimensional systems than what has been done in this thesis, promoting sparsity will presumably be vital for achieving accuracy.

6.6 Analysis and discussion

After studying experiments on four different systems in this chapter, we can infer that the PHSI model is largely prosperous. In sections 6.1 and 6.2, we see that the model works well on pure Hamiltonian systems. Figures 6.1 and 6.2 illustrate that the model predicts well on noisy data, indicating that the integration scheme is successful. Figure 6.3 illustrates that the PHSI model's pseudo-Hamiltonian structure helps keep the model prediction stable as opposed to the predictions of the BSI and SINDy models that diverge when trained on noisy data. The PHSI model also learned the Hamiltonian equations through system identifications more accurately than the other models. Its Hamiltonian structure helps narrow the function dictionary in high dimensional systems, and Figure 6.4 suggests that this enhances its system identification ability. In the mass-spring experiment, the PHSI model was able to separately learn the inner dynamics, damping, and external forces accurately through separate system identification models. This allowed for full insight into the system's behavior as well as excellent prediction outside of the domain of training data. In the tank experiment, the PHSI model accurately retrieved the true equations for the inner dynamics and the damping despite the complex form of the external forces. Since the other models mostly failed to do so, we can infer that the separation of inner and outer dynamics is crucial for system identification in these experiments.

We also note that the PHSI model has limitations. Firstly, the model assumes that the structure matrix in the Hamiltonian formulation (see Equation 2.26) is known in all the numerical experiments. This restricts the model to learning systems where the structure matrix is prior knowledge, a restriction that other system identification models such as SINDy [BPK16] do not have. However, in some cases, the structure matrix can be obtained from physical intuition and engineering knowledge (e.g. the tank problem). Secondly, since the separation of the inner and outer dynamics in the model is non-unique, we have no guarantee of finding the separation that most accurately reflects the true system composition. As discussed in Section 5.2, the model user may have to impose prior physical knowledge to achieve the most natural separation of the inner and outer dynamics. In addition, the PHSI model assumes that the location of the external forces is known, i.e. which dimensions in the system the forces affect. [Eid+23] show that the performance of their pseudo-Hamiltonian neural networks declines when the location of the external forces is unknown. This weakness may affect the PHSI model as well, although we have not conducted this experiment. Also, a general limitation of system identification is that we have to guess what functions to include in the search dictionary, and it can be hard to know these function forms beforehand. Yet, in practical cases, many systems can be described by polynomials, meaning the PHSI is applicable to a wide range of physical systems.

CHAPTER 7

Conclusions

In this thesis, we have combined theory from Hamiltonian mechanics and system identification to propose a novel way of learning dynamical systems from data. We reviewed Lagrangian and Hamiltonian mechanics as well as machine learning techniques including neural networks and system identification.

Using this theory, we wanted to assess the application of system identification to realistic dynamical systems affected by damping and external forces. Our study introduced a pseudo-Hamiltonian system identification model (PHSI), and the results of several numerical experiments demonstrated that this is feasible. However, the PHSI model's success relies on several prior physical assumptions. We also wanted to investigate whether it is possible to deploy separate system identification models to learn the inner dynamics, damping, and external effects all at once. This was also accomplished by the PHSI model in a numerical experiment, albeit on a simple, low-dimensional system.

The PHSI model outperformed existing system identification models on Hamiltonian and pseudo-Hamiltonian systems. It also holds an advantage in terms of interpretability over black-box dynamical system models. We introduced a training algorithm based on a numerical integration scheme that proved effective, especially on noisy data. We also addressed promoting sparsity using regularization and pruning, and we conducted a test of hyper-parameters. As discussed in Section 7, the method of the PHSI model has some limitations and weaknesses. Some examples are requirements of prior physical assumptions and non-uniqueness of the pseudo-Hamiltonian formulation. Yet, the PHSI model provides a promising approach to system identification that can be applied in various fields.

Bibliography

[BL07] Bongard, J. and Lipson, H. 'Automated reverse engineering of nonlinear dynamical systems'. In: *Proceedings of the National Academy of Sciences* vol. 104, no. 24 (2007), pp. 9943–9948.

[BPK16] Brunton, S. L., Proctor, J. L. and Kutz, J. N. 'Discovering governing equations from data by sparse identification of nonlinear dynamical systems'. In: *Proceedings of the national academy of sciences* vol. 113, no. 15 (2016), pp. 3932–3937.

[Che+20] Chen, Z. et al. 'Symplectic Recurrent Neural Networks'. In: *International Conference on Learning Representations*. 2020.

[Cra+20] Cranmer, M. et al. 'Lagrangian Neural Networks'. In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations* (2020).

[Cyb89] Cybenko, G. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of control, signals and systems* vol. 2, no. 4 (1989), pp. 303–314.

[DA21] Duong, T. and Atanasov, N. 'Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control'. In: *Robotics: Science and Systems (RSS)*. 2021.

[Dau+10] Daubechies, I. et al. 'Iteratively reweighted least squares minimization for sparse recovery'. In: *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences* vol. 63, no. 1 (2010), pp. 1–38.

[DOY11] Dahlby, M., Owren, B. and Yaguchi, T. 'Preserving multiple first integrals by discrete gradients'. In: *Journal of Physics A: Mathematical and Theoretical* vol. 44, no. 30 (2011), p. 305205.

[Du+19] Du, S. et al. 'Gradient descent finds global minima of deep neural networks'. In: *International conference on machine learning*. PMLR. 2019, pp. 1675–1685.

[DXZ20] DiPietro, D., Xiong, S. and Zhu, B. 'Sparse symplectically integrated neural networks'. In: *Advances in Neural Information Processing Systems* vol. 33 (2020), pp. 6074–6085.

[Eid+23] Eidnes, S. et al. 'Pseudo-Hamiltonian neural networks with state-dependent external forces'. In: *Physica D: Nonlinear Phenomena* vol. 446 (2023), p. 133673.

[Eva22] Evans, L. C. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[Fou22] Foucart, S. *Mathematical pictures at a Data Science Exhibition*. Cambridge University Press, 2022.

[FR13] Foucart, S. and Rauhut, H. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser New York, 2013.

[GBC16] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. MIT press, 2016.

[GDY19] Greydanus, S., Dzamba, M. and Yosinski, J. 'Hamiltonian Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by Wallach, H. et al. Vol. 32. Curran Associates, Inc., 2019.

[HER23] Holmsen, S., Eidnes, S. and Riemer-Sørensen, S. 'Pseudo-Hamiltonian System Identification'. In: *arXiv preprint arXiv:2305.06920* (2023).

[HH64] Hénon, M. and Heiles, C. 'The applicability of the third integral of motion: Some numerical experiments'. In: *Astronom. J.* vol. 69 (1964), pp. 73–79.

[Hor91] Hornik, K. 'Approximation capabilities of multilayer feedforward networks'. In: *Neural networks* vol. 4, no. 2 (1991), pp. 251–257.

[Jin+20] Jin, P. et al. 'SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems'. In: *Neural Networks* vol. 132 (2020), pp. 166–179.

[JJL98] Jürgen Jost, J., Jost, J. and Li-Jost, X. *Calculus of variations*. Vol. 64. Cambridge University Press, 1998, p. 6.

[Kar+21] Karniadakis, G. E. et al. 'Physics-informed machine learning'. In: *Nature Reviews Physics* vol. 3, no. 6 (2021), pp. 422–440.

[KB14] Kingma, D. P. and Ba, J. 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980* (2014).

[Koz94] Koza, J. R. 'Genetic programming as a means for programming computers by natural selection'. In: *Statistics and computing* vol. 4, no. 2 (1994), pp. 87–112.

[Les+93] Leshno, M. et al. 'Multilayer feedforward networks with a non-polynomial activation function can approximate any function'. In: *Neural networks* vol. 6, no. 6 (1993), pp. 861–867.

[Lju87] Ljung, L. *System identification: theory for the user*. Prentice Hall Information and System Sciences Series. Prentice Hall, Inc., Englewood Cliffs, NJ, 1987, pp. xxi+519.

[LTS22] Lee, K., Trask, N. and Stinis, P. 'Structure-preserving sparse identification of nonlinear dynamics for data-driven modeling'. In: *Proceedings of Mathematical and Scientific Machine Learning* vol. 190 (2022), pp. 65–80.

[Ma+19] Ma, R. et al. 'Transformed 1 regularization for learning sparse deep neural networks'. In: *Neural Networks* vol. 119 (2019), pp. 286–298.

[MIY20] Matsubara, T., Ishikawa, A. and Yaguchi, T. 'Deep Energy-based Modeling of Discrete-Time Physics'. In: *Advances in Neural Information Processing Systems*. Ed. by Larochelle, H. et al. Vol. 33. Curran Associates, Inc., 2020, pp. 13100–13111.

[MQR99] McLachlan, R. I., Quispel, G. R. W. and Robidoux, N. 'Geometric integration using discrete gradients'. In: *R. Soc. Lond. Philos. Trans. Ser. A Math. Phys. Eng. Sci.* vol. 357, no. 1754 (1999), pp. 1021–1045.

[MR99] Marsden, J. E. and Ratiu, T. S. *Introduction to mechanics and symmetry*. Second. Vol. 17. Texts in Applied Mathematics. A basic exposition of classical mechanical systems. Springer-Verlag, New York, 1999, pp. xviii+582.

[Nat95] Natarajan, B. K. 'Sparse approximate solutions to linear systems'. In: *SIAM journal on computing* vol. 24, no. 2 (1995), pp. 227–234.

[RPK19] Raissi, M., Perdikaris, P. and Karniadakis, G. E. 'Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations'. In: *J. Comput. Phys.* vol. 378 (2019), pp. 686–707.

[SL09] Schmidt, M. and Lipson, H. 'Distilling free-form natural laws from experimental data'. In: *science* vol. 324, no. 5923 (2009), pp. 81–85.

[Tao16] Tao, M. 'Explicit symplectic approximation of nonseparable Hamiltonians: Algorithm and long time performance'. In: *Physical Review E* vol. 94, no. 4 (2016), p. 043303.

[Tib96] Tibshirani, R. 'Regression shrinkage and selection via the lasso'. In: *Journal of the Royal Statistical Society: Series B (Methodological)* vol. 58, no. 1 (1996), pp. 267–288.

[Van06] Van Der Schaft, A. 'Port-Hamiltonian systems: an introductory survey'. In: *Proceedings of the international congress of mathematicians*. Vol. 3. Citeseer. 2006, pp. 1339–1365.

[VJ14] Van Der Schaft, A. and Jeltsema, D. 'Port-Hamiltonian systems theory: An introductory overview'. In: *Foundations and Trends in Systems and Control* vol. 1, no. 2-3 (2014), pp. 173–378.

[Wil+22] Willard, J. et al. 'Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems'. In: *ACM Comput. Surv.* vol. 55, no. 4 (Nov. 2022).

[Zhu+22] Zhu, A. et al. 'On numerical integration in neural ordinary differential equations'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 27527–27547.

Appendices

APPENDIX A

Numerical comparison of integration schemes

In Section 5.1, we discussed the choice of numerical integrator Ψ when computing the loss for the PHSI model (Equation 5.6). In all the experiments in Chapter 6, the choice fell on the symmetric fourth-order Runge-Kutta method introduced by [Eid+23], and it was argued that this integrator handles noise well. In this appendix, we will compare the performance of this integrator with other numerical integrators. More specifically, we will experiment with training the PHSI models with the following integrators:

- the forward Euler method (Euler), (Equation 5.7)
- the implicit midpoint method (Midpoint), (Equation 5.8)
- the classic fourth-order Runge–Kutta method (RK4), (Equation 5.9)
- the symmetric fourth-order Runge–Kutta method (SRK4), (Equation 5.11)

and their properties are summarized in Table A.1.

Table A.1: Properties of integrators.

Integrator	order	num. evalulations	explicit	symmetric
Forward Euler (Euler)	1	1	yes	no
Implicit midpoint (Midpoint)	2	1	no	yes
Runge–Kutta-4 (RK4)	4	4	yes	no
Symmetric 4th order (SRK4)	4	4	no	yes

Note that the fourth-order methods require multiple evaluations of the PHSI model, making them more computationally expensive. Note also that although the midpoint and SKR4 methods are implicit, they can be explicitly computed in the integration scheme as explained in Section 4.3.

A.1 Hamiltonian system: Hénon–Heiles

We train the models on trajectories from $t = 0$ to $t = 1$ with sampling time 0.1 and

- 2000 samples, i.e. 200 trajectories, with $\sigma = 10^{-5}$, i.e. nearly without noise;
- 2000 samples with small noise (Gaussian noise with a standard deviation $\sigma = 0.001$ added to the measurements of the states);
- 2000 samples with moderate noise (standard deviation $\sigma = 0.01$);

When trained on data without noise, the PHSI model performs better using the fourth-order integrators (RK4, SRK4), as seen in Figure A.1. The error when using the Euler method is two orders of magnitude larger than for the SRK4 method. The same can be observed when noise is added, but the difference becomes smaller. In general, the forward Euler method that only relies on one data point in the evaluation of Ψ gives a lower accuracy than the other methods that rely on two data points. The accuracy of the PHSI model seems to be less affected by the choice of integrator when the dataset contains a large amount of noise.

Our approach in the integration scheme is equivalent to performing one integration step. [DXZ20] apply an integration scheme where several steps are performed, increasing the accuracy of their results. However, performing multiple steps will add a computational cost, and it is not proven that it will increase the accuracy on data with noise.

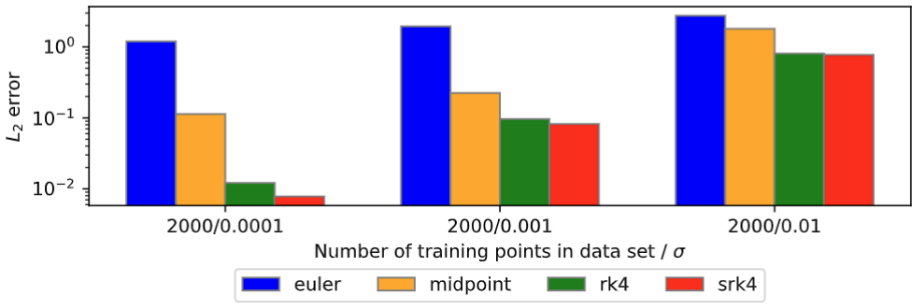


Figure A.1: The mean l_2 error of PHSI models trained with the different integrators on the Hénon–Heiles problem. The error is on the predicted positions and momenta from $t = 0$ to $t = 1$ on 20 different random initial conditions.

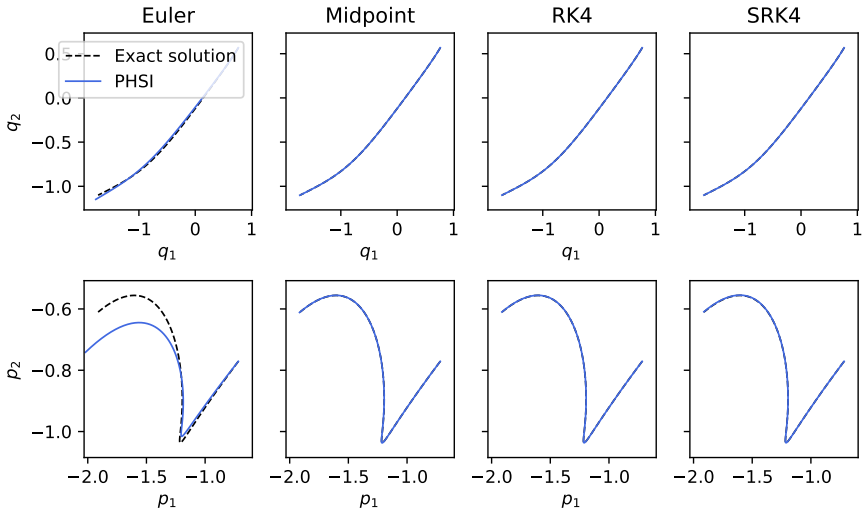


Figure A.2: Prediction of the momentum and position by PHSI models trained with different integrators on the Hénon–Heiles problem.

A.2 Pseudo-Hamiltonian system: Tanks and pipes

We also test the performance of the different integrators in the hybrid model used to learn the tank system presented in Section 6.4. We train the models on trajectories from $t = 0$ to $t = 0.5$ with sampling time 0.01 and

- 3000 samples, i.e. 60 trajectories, without noise;
- 3000 samples with moderate noise (Gaussian noise with a standard deviation $\sigma = 0.01$ added to the measurements of the states);
- 3000 samples with much noise (standard deviation $\sigma = 0.05$);

Figure A.3 shows the l_2 error of all the state variables from the predictions obtained by applying the methods on a test set of 10 different initial conditions. The forward Euler method performs significantly worse than the other methods in all the experiments, something that is not surprising considering it is only of first order and only depends on one data point in the evaluation of Ψ. The fourth-order methods perform better than the lower-order methods for all the experiments, illustrating that using higher-order integrators may be worth the extra computational cost. We also observe that the symmetric methods perform well on the dataset with much noise, supporting the claim that symmetry in the integrator evens out the noise between the data points. As seen in Figure A.4, the choice of integrator is vital to the PHSI model's predictive abilities. SRK4 outperforms RK4 and the midpoint methods, which again outperform the forward Euler method.

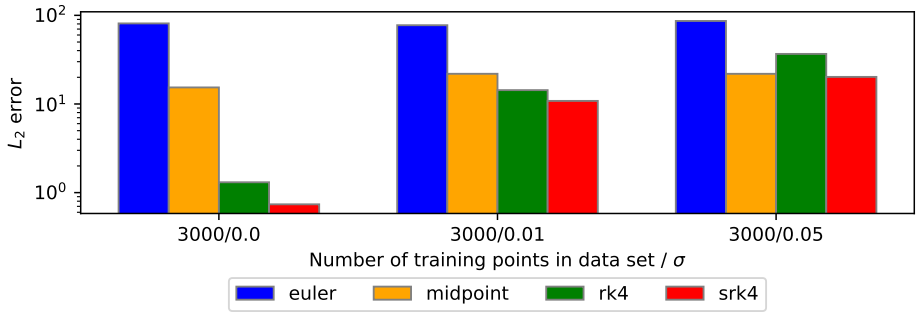


Figure A.3: The mean l_2 error of hybrid PHSI models trained with the different integrators on the tank system. The error is of the predicted volume and flow in all tanks and pipes from $t = 0$ to $t = 0.5$ on 10 different random initial conditions.

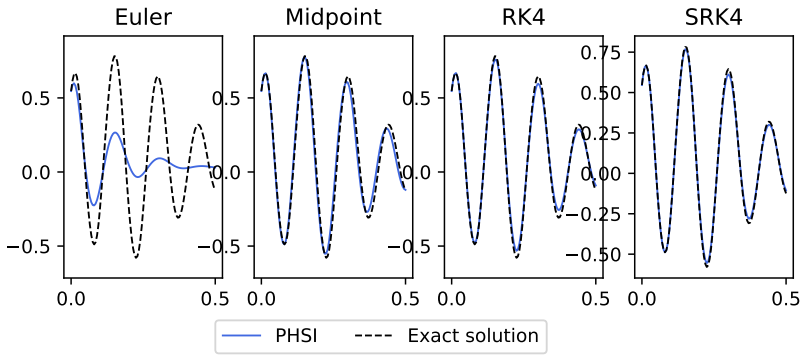


Figure A.4: Volume of the fourth tank as predicted by hybrid PHSI models trained with different integrators on the tank system.

APPENDIX B

Implementation details

For all experiments of this paper, the Adam optimizer is used, with a weight decay constant of 10^{-4}. The learning rate is chosen from the following search space: $\{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}\}$. The training data is shuffled for all experiments, and the integrator is the fourth-order symmetric integration scheme introduced in [Eid+23]. The batch size is 32. For the PHSI and BSI models, polynomial coefficients have an initial value of 0.2, and trigonometric coefficients (amplitude and frequency) have an initial value of 1.

B.1 Hénon–Heiles system

In this experiment, the learning rate was $3 \cdot 10^{-3}$. Regularization was not used. The PHSI and BSI models were trained for 60 epochs. The P-value and ϵ-value in Algorithm 6 were set to 5 and 0.05, respectively. The noisy data set had $\sigma = 0.02$. The search space for PHSI and SSINN was third-degree polynomials, while for SINDy and BSI it was second-degree polynomials. The initial state values in the training data are drawn uniformly between -1 and 1.

B.2 Nonlinear Schrödinger system

For the PHSI model, the learning rate was chosen to be 10^{-2}. Regularization was not used. The model was trained for 100 epochs. The P-value and ϵ-value in Algorithm 6 were set to 20 and 0.05, respectively. The noisy dataset had $\sigma = 5 \cdot 10^{-5}$. The search space for PHSI was fourth-degree polynomials, while for SINDy and BSI it was third-degree polynomials. The initial state values in the training data are drawn uniformly between -1 and 1.

B.3 Forced and damped mass-spring system

The SINDy model has a disadvantage as it can only learn autonomous systems. To still be able to use it, we converted the system into an autonomous one by adding time as a variable to Equation 6.5: $\dot{t} = 1$.

The PHSI, BSI, and PHNN models were run for 150 epochs, with the learning rate $3 \cdot 10^{-3}$. The P-value and ϵ-value in Algorithm 6 were set to 20 and 0.05, respectively. l_1-regularization was used for the PHSI model on the forces and the Hamiltonian function: $\lambda_1 = 0.1$ and $\lambda_2 = 0.01$ in Equation 5.6. The port of

the PHNN had a regularization parameter of 0.1. In the PHSI model, $\hat{H}$ had a search space of polynomials of the third degree, and $\hat{F}$ had a combination of polynomial and trigonometric terms and was strictly dependant on the state q, p. The BSI and SINDy models trained on a combination of polynomial terms of third-degree and trigonometric terms. The initial state values in the training data are drawn uniformly between -1 and 1.

B.4 System of tanks and pipes

The learning rate $3 \cdot 10^{-2}$ was chosen. The PHSI and BSI models were trained for 100 epochs. The P-value and ϵ-value in Algorithm 6 were set to 10 and 0.05, respectively. For the PHSI model, l_1 regularization was used on the external forces and the Hamiltonian with $\lambda_1 = 0.5, \lambda_2 = 0.001$ in Equation 5.6. In the PHSI model, the external forces were modeled by a neural network with three hidden layers of 100 nodes. The search space for $\hat{H}$ was second-degree polynomials. The BSI and SINDy models had a search space of first-degree polynomials. The initial state values in the training data are drawn uniformly between -1 and 1.