SympNets, PNNs and GFINNs: Intrinsic structure preserving neural networks for identifying and solving dynamical systems with applications to optimal control problems

by

Zhen Zhang

B.Sc., City University of Hong Kong, Hong Kong, P. R. China, 2018

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Division of Applied Mathematics at Brown University

PROVIDENCE, RHODE ISLAND

May 2024

This dissertation by Zhen Zhang is accepted in its present form
by the Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date_____ _____

George Em Karniadakis, Ph.D., Advisor

Recommended to the Graduate Council

Date_____ _____

Jérôme Darbon, Ph.D., Co-Advisor

Date_____ _____

Brendan Keith, Ph.D., Reader

Approved by the Graduate Council

Date_____ _____

Thomas A. Lewis, Dean of the Graduate School

Vita

Zhen Zhang was born to Qinhui Zhang and Shujun Zhang in 1996 in Nantong, Jiangsu Province, China. He completed his Bachelor of Science degree in computational mathematics in June 2018 at City University of Hong Kong in Hong Kong, China. He then moved to Providence, Rhode Island, to pursue a doctoral degree in the Division of Applied Mathematics at Brown University in August 2018, where he obtained a Master of Science degree in applied mathematics in 2019. Ever since middle school, Zhen is determined to major in mathematics for his undergraduate study. At City University of Hong Kong, he was given the opportunity to lead an undergraduate research project in his senior year under the supervisory of Professor Xiang-Sheng Wang and Professor Roderick Wong on the asymptotic analysis of Painlevé equations. He then found stronger interest in the combination of deep learning and applied mathematics during his bachelor's thesis advised by Professor Ding-Xuan Zhou on the structural analysis of deep convolution neural networks. At Brown University, Zhen worked under the mentorship of Professor George Em Karniadakis, who is a renowned scientist in the field of computational fluid dynamics and applied mathematics. He got the opportunity to work with researchers across the globe to develop state-of-the-art neural network models which obeys physical laws including symplecticity and GENERIC formalism. In addition, Zhen has been a teaching assistant for two introductory courses in applied mathematics: Introduction to Computational Linear Algebra (Fall 2019) and Statistical Inference II (Spring 2020). He has also been actively participating in conference presentations, including SIAM MDS22, CSE23.

Acknowledgments

Firstly, I would like to thank my advisor, Professor George Em Karniadakis, for his support and supervision during my Ph.D. study at Brown University. He is not only an exceptional mentor but also possesses a keen intellect and tireless dedication to his students. I am immensely grateful for the opportunity to benefit from his wisdom and passion for research. His readiness to engage with new discoveries and address any queries I may have regarding my research projects has been invaluable. Being a part of the CRUNCH group has been a source of immense pride, and it's fortunate for me to have him as both a teacher and mentor. I would like to express my sincere gratitude to my dissertation committee for dedicating their time to review my dissertation and for providing invaluable feedback and suggestions. I am particularly thankful to Professor Jérôme Darbon for his insightful guidance on symplectic optimal control networks, which greatly influenced the development of my thesis.

I extend special thanks to my collaborator, Dr. Pengzhan Jin, for his invaluable assistance during my initial research project on the theoretical analysis of symplectic networks. I am also grateful to Professor Yeonjong Shin for his close collaboration on GENERIC formalism informed neural networks, which proved to be a source of motivation during the most challenging phases of my Ph.D. studies. I have benefited immensely from the expertise and guidance of Professor Guang Lin and Dr. Sheng Zhang in the area of uncertainty quantification. Dr. Tingwei Meng and Dr. Shanqing Liu are great experts in optimal control and Hamilton-Jacobi theory. Their

innovative ideas and dedication to research have been a great motivation for me. I also wish to extend my thanks to Alan John Varghese, Vivek Oommen and Qian Zhang whose discussions and implementations have sparked fresh ideas regarding the design of novel neural network architectures that I have never thought of before. Additionally, I appreciate the dedication of Professor Hongjie Dong, Professor Chi-Wang Shu, Professor Walter Strauss, and Professor Matthew Harrison for their teaching and support during my first year, as well as for their roles in my preliminary exams committee. Their commitment has been pivotal in establishing a strong mathematical foundation that has shaped my thesis.

I extend heartfelt appreciation to my dear friends, both near and far. Han You, Bingze Xu, Shentao Yang, and Zhongjie Shi in the United States, as well as Zhongjie Shi in China, hold a special place in my heart. I fondly recall the joyous gatherings with Enrui Zhang, Xiaoyu Xie, Sicheng Liu, Xin Lian, Qian Zhang, Yi Jiang, Tianming Yu, Xiaoding Yang, Yue Wu and Yidi Wu during the final year of my graduate studies. I am also deeply grateful to all current and former colleagues within our group, whose camaraderie and support have been invaluable. Among them are Aniruddha Bora, Min Cai, Shengze Cai, Zhaojie Chai, Xiaoli Chen, Yixiang Deng, Maximilian Dreisbach, Mario De Florio, Somdatta Goswami, Xiaowei Jin, Adar Kahana, Ehsan Kharazmi, Alena Kopanicakova, Varun Kumar, Youngkyu Lee, Guansheng Li, He Li, Xiang Li, Zhen Li, Chensen Lin, Zixiang Liu, Lu Lu, Zhiping Mao, Sathesh Mariappan, Xuhui Meng, Kamaljyoti Nath, Guofei Pang, Ahmad Peyvan, Jens Peter Schoeler, Khemraj Shukla, Juan Diego Toscano, Ruyin Wan, Chenye Wang, Shupeng Wang, Zhibo Wang, Zhicheng Wang, Chenxi Wu, Mengjia Xu, Liu Yang, Minglang Yin, Dongkun Zhang, Xiaoning Zheng, Qiang Zheng, Zongren Zou, and many others. Each of you has contributed to the unforgettable moments we've shared together, and for that, I am truly thankful.

Last but not least, I have my utmost gratitude towards my mother Shujun Zhang and my father Qinhui Zhang. Without their encouragement and trust, I would not be able to accomplish my doctoral study with such a strong determination and confidence. I am thankful to their unconditional love and always being supportive to my decisions. Lastly, I would like to thank my wife, Mengjie Liu. Her company and support are the main reasons I have survived this strenuous Ph.D. journey.

Publication

* *indicates equal contribution or alphabetical order.*

1. Pengzhan Jin*, **Zhen Zhang***, Aiqing Zhu, Yifa Tang and George Em Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. Neural Networks 132, p. 166-179, 2020.

2. Ehsan Kharazmi, Min Cai, Xiaoning Zheng, **Zhen Zhang**, Guang Lin and George Em Karniadakis. Identifiability and predictability of integer- and fractional-order epidemiological models using physics-informed neural networks. *Nature Computational Science*, 2021.

3. Sheng Zhang*, Joan Ponce*, **Zhen Zhang***, Guang Lin and George, Karniadakis. An integrated framework for building trustworthy data-driven epidemiological models: Application to the COVID-19 outbreak in New York City. *PLOS Comput Biol* 17(9), 2021.

4. Pengzhan Jin, **Zhen Zhang**, Yannis Kevrekidis and George Em Karniadakis. Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2022.

5. **Zhen Zhang**, Yeonjong Shin and George Em Karniadakis. GFINNs: GENERIC Formalism Informed Neural Networks for Deterministic and Stochastic Dynamical Systems. Philos. Trans. R. Soc. A, 2022.

6. Tingwei Meng*, **Zhen Zhang***, Jerome Darbon and George Em Karniadakis. SympOCnet: Solving optimal control problems with applications to high-dimensional multi-agent path planning problems. *SIAM Journal on Scientific Computing* 44(6), 2022.

7. Mitchell Daneker, **Zhen Zhang**, George Em Karniadakis, Lu Lu. Systems Biology: Identifiability analysis and parameter identification via systems-biology informed neural networks.*Methods in Molecular Biology*, 2023.

8. **Zhen Zhang**, Zongren Zou, Ellen Kuhl and George Em Karniadakis. Discovering a reaction-diffusion model for Alzheimer's disease by combining PINNs with symbolic regression. *Computer Methods in Applied Mechanics and Engineering* 2024.

9. Nick-Marios T. Kokolakis, **Zhen Zhang**, Kyriakos G. Vamvoudakis, Jérôme Darbon, and George Em. Karniadakis. Safe Physics-Informed Machine Learning for Optimal Predefined-Time Stabilization: A Lyapunov-based Approach. (Submitted to *IEEE Transactions on Neural Networks and Learning Systems*)

10. **Zhen Zhang**, Chenye Wang, Shanqing Liu, Jérôme Darbon, and George Em. Karniadakis. TSympOCNet for nonconvex path planning problems with linear and nonlinear dynamics. (In preparation)

Abstract of "SympNets, PNNs and GFINNs: Intrinsic structure preserving neural networks for identifying and solving dynamical systems with applications to optimal control problems", by Zhen Zhang, Ph.D., Brown University, May 2024

This thesis introduces novel neural network architectures designed to discover dynamical systems from physical data and as well as solve dynamical systems rising from optimal control problems. At the core of the investigation are three interconnected neural network models: Symplectic Networks (SympNets), Poisson Neural Networks (PNNs), and GENERIC Formalism Informed Neural Networks (GFINNs).

SympNets are introduced as a foundational architecture to approximate arbitrary symplectic maps, based on our proof of the universal approximation theorem. We apply SympNets and its variants in three different scenarios, (i) identifying Hamiltonian systems from data, (ii) node classficiation on graphs (SympGNNs), and (iii) solve high-dimensional optimal control problems through a symplectic transformation (SympOCNets and TSympOCNets).

Building on SympNets, we further introduce PNNs to extend the framework to Poisson systems, addressing noncanonical coordinates through the integration of the Darboux-Lie theorem. GFINNs represent a further generalization, designed to encompass noncanonical conservative and dissipative systems by incorporating the GENERIC formalism. These models significantly enhance the capability of SympNets to model more complex dynamics arising from a broader spectrum.

Collectively, these models represent an advancement in the integration of machine learning with physical modeling, potentially offering new ways to solve problems in system identification and optimal control. The thesis not only establishes new theoretical results but also provides empirical evidence of the models' superior performance in a variety of complex scenarios, paving the way for future research in physics informed machine learning.

Contents

List of Tables

List of Figures

Chapter One

Introduction

The discovery of governing equations for dynamical systems from observed data is a longstanding scientific endeavor [19, 20, 173]. The so-called data-driven discovery dated back to Kepler refers to scientific methods that extract important features from data and either approximate or identify governing equations by means of (parameterized) function classes. With the recent advancement in deep learning, neural network classes have been popularly employed in modelling and simulations and have demonstrated some promising empirical results [22, 57, 83, 132, 156, 157].

The data-driven discovery may be classified into two major approaches. One is the pure data-driven methods[31, 106] which model governing equations using neural networks that are trained to fit observed data *without physics*. This approach could provide a neural network model that mimics training trajectories, particularly when no physics but a large amount of data are available. However, it is likely that the learned models do not generalize well on the region where training data are scarce or even do not exist. The other is the physics-informed data-driven approach, which aims to model governing equations by *embedding principles of physics into neural networks* together with data. By exploiting physics, it was empirically observed that the amount of data needed to get good performance is much less than those of the pure data-driven methods, and the learned model is stable and generalizes well. Typically, the physics are imposed on neural networks by means of either soft or hard constraints. The use of soft constraints introduces regularization terms in an associated loss function that penalize neural networks that do not obey the physics [158, 87]. The hope is that neural networks approximately follow the physics after training. The hard constraints are typically imposed by designing proper neural network architectures that obey the underlying principles without optimization processes [81, 101, 102, 35], yet maintain sufficient expressivity so that governing equations can be learned from data. This is the approach we follow in this thesis.

A schematic diagram of the classification of the data-driven discovery is given in Figure 1.1.

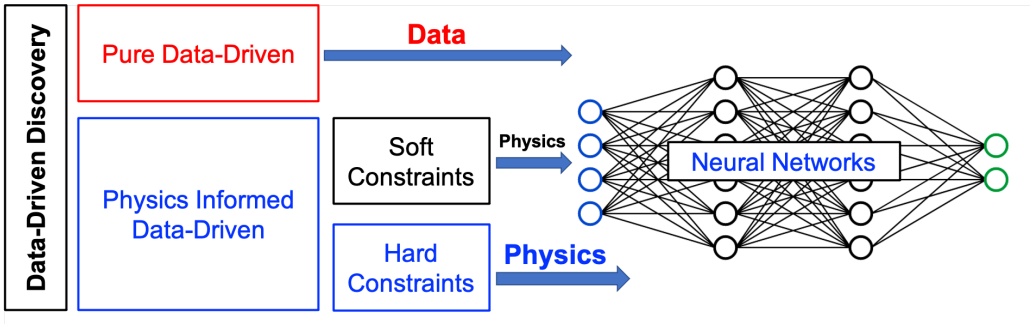


Figure 1.1: A schematic diagram showing the major three approaches in the data-driven discovery of dynamical systems. One is the pure data-driven approach. The other two are the physics informed data-driven approach. The second one is based on soft constraints imposed in an associated loss function. The third one directly imposes the underlying physics into neural networks by designing novel network architectures.

In this thesis, we present three different neural networks with physical principles embedded directly into the architectures. Our target is to identify a map $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ such that it satisfies one or more of the following principles:

1. **Symplecticity**

$$\left(\frac{\partial \Phi}{\partial x}\right)^\top J \left(\frac{\partial \Phi}{\partial x}\right) = J,$$

where $J := \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}$.

2. **Poisson**

$$\left(\frac{\partial \Phi}{\partial x}\right) B(x) \left(\frac{\partial \Phi}{\partial x}\right)^\top = B(\Phi(x)),$$

where B is anti-commutative, bilinear and satisfies Leibniz's rule.

3. **GENERIC formalism**

$$\begin{aligned} & \Phi(x) = x + L(x)\frac{\partial E}{\partial x}(x) + M(x)\frac{\partial S}{\partial x}(x), \\ \text{subject to} \quad & L(x)\frac{\partial S}{\partial x}(x) = M(x)\frac{\partial E}{\partial x}(x) = \mathbf{0}, \\ & L(x) \text{ is skew-symmetric, i.e., } L(x) = -L(x)^\top \\ & M(x) \text{ is symmetric positive semi-definite.} \end{aligned}$$

4. **Permutation equivariance**

$$\Phi(Px) = P\Phi(x),$$

where P is a permutation matrix.

The neural networks developed and discussed are symplectic networks (SympNets), Poisson neural networks (PNNs), and GENERIC formalism informed neural networks (GFINNs), each designed to be a universal approximator for functions adhering to the first three principles, respectively. Additionally, we present Symplectic Graph Neural Networks (SympGNNs), which satisfy both the symplecticity and permutation equivariance principles, aimed at identifying high-dimensional Hamiltonian systems and performing node classifications.

These networks are constructed to inherently guide themselves towards accurate solutions and demonstrate superior generalization capabilities compared to purely data-driven models. This performance advantage is particularly noticeable when the underlying data reflects the same physical principles and when the number of training examples is limited.

Apart from data-driven discovery tasks, another specific application of Symp-

Nets is on the optimal control problems. Due to the Pontryagin's maximum principle, control problems can be reformulated to solve Hamiltonian ordinary differential equations (ODEs). By employing a SympNet, we can pinpoint a symplectic transformation that simplifies the original ODE into an easier-to-solve latent format. Following this, we revert to the original phase space using the SympNet's inverse function to retrieve the solution. Our experiments demonstrate that such a transformation is universally applicable across a diverse spectrum of scenarios, ranging from linear to nonlinear dynamics and from single to double integrators. This methodology has been successfully applied to multi-agent path planning challenges, proving its capability and efficiency in tackling high-dimensional tasks. Notably, it enabled the resolution of a path planning issue for 256 drones in just 1.5 hours.

The rest of the thesis is organized as follows: In Chapter 2, we present the SympNets (SympGNNs) and their applications to the data-driven discovery of Hamiltonian systems. In Chapter 3, we present the generalization of SympNets to noncarnonical coordinates, i.e. PNNs, through the integration of Darboux-Lie theorem. In Chapter 4, we propose GFINNs to identify both conservative and dissipative systems. In Chapter 5 and 6, we apply SympNets to optimal control problems and introduce SympOCNets and TSympOCNets.

Chapter Two

SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems

2.1 Introduction

It is well known that neural networks can approximate continuous maps [36, 90]. However, universal approximation theorems only guarantee a small approximation error for a sufficiently large network, but do not consider the optimization and generalization errors. In order to obtain satisfactory results for a given task, big data is required that we may not be able to afford if this task is regarded as a pure approximation problem [98]. For this reason, when applying deep learning to physical systems, the cost of data acquisition is prohibitive, and we are inevitably faced with the challenge of drawing conclusions and making decisions under partial information. Fortunately, for physical systems there exists a vast amount of prior knowledge that is not always utilized in machine learning practices. Encoding such structured information into a learning algorithm results in amplifying substantially the information content of the data that the algorithm sees, enabling it to quickly steer itself towards the right solution and to generalize well even when only a few training examples are available [117, 158]. There have been many research works focusing on how to employ prior knowledge to construct the targeted machine learning algorithms for specific problems, where the approximated maps usually have special structures or properties, which we naturally expect the trained networks to possess, such as image classification [113], natural language processing [134], game playing [178], as well as the recent work [129] providing a special network structure based on the universal approximation theorem for approximating nonlinear operators [30]. Some works enforce the prior information on network structures by manifold calculus [68, 69, 70]. Additionally, more contributions specific to solving problems on the manifold of symplectic matrices were proposed in [71, 72, 189].

In this chapter, we aim to study how to impose the prior information on the neural

networks for identifying Hamiltonian systems. Specifically, we focus on endowing the neural networks with a symplectic structure.

First, we provide some relevant background material. Denote the d-by-d identity matrix by I_d, and let

$$J := \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix},$$

which is an orthogonal, skew-symmetric real matrix, so that $J^{-1} = J^T = -J$.

Definition 1. A matrix $H \in \mathbb{R}^{2d\times 2d}$ is called symplectic if $H^T J H = J$.

With the concept of symplectic matrix, the definition of symplectic map can be given.

Definition 2. A differentiable map $\Phi : U \to \mathbb{R}^{2d}$ (where $U \subset \mathbb{R}^{2d}$ is an open set) is called symplectic if the Jacobian matrix $\frac{\partial \Phi}{\partial x}$ is everywhere symplectic, i.e.,

$$\left(\frac{\partial \Phi}{\partial x}\right)^{\top} J \left(\frac{\partial \Phi}{\partial x}\right) = J.$$

We consider the Hamiltonian system

$$\begin{cases} \dot{y} = J^{-1}\nabla H(y) \\ y(t_0) = y_0 \end{cases}, \tag{2.1}$$

where $y(t) \in \mathbb{R}^{2d}$, and H is the Hamiltonian typically representing the energy of the system (2.1). Let $\phi_t(y_0)$ be the phase flow of system (2.1). In 1899, Poincaré pointed out that the phase flow of a Hamiltonian system is a symplectic map [84, p. 184, Theorem 2.4], i.e.,

$$\left(\frac{\partial \phi_t}{\partial y_0}\right)^{\top} J \left(\frac{\partial \phi_t}{\partial y_0}\right) = J. \tag{2.2}$$

The evaluation of the behavior of dynamical systems at long time is a notoriously difficult problem in mathematics, particularly for discrete dynamical systems. One may encounter situations where the dynamics explodes, converges to stationary states or exhibits chaotic behavior. Fortunately, for Hamiltonian systems, these problems can be alleviated by imposing the symplectic structure on the numerical methods due to (2.2). There are some well-developed works on symplectic integration, see for example [65, 84, 133]. As the symplectic numerical integrators yield transformative results across diverse applications based on the Hamiltonian systems [147, 64, 194, 155], we aim to consider the construction of networks possessing symplecticity and explore how it impacts the numerical methods for the Hamiltonian systems.

To this end, many neural network-based models have been proposed to identify the Hamiltonian systems from data [13, 81, 162, 172, 32, 184, 196], with further applications in image prediction [81], generative modeling [184] and continuous control [196]. These learning models are mostly constructed by exploiting the structure of standard numerical time-stepping methods [80, 29, 157]. The most fundamental learning model specific to Hamitonian systems is proposed in [81] named Hamiltonian neural networks (HNNs), which uses a standard neural network $\widetilde{H}$ to approximate the Hamiltonian H instead of the total vector field $J^{-1}\nabla H$. The input to HNNs are the phase points as well as their derivatives. If only time-dependent discrete phase points are available, a numerical integrator has to be applied to the data to construct the loss. A subsequent work in [32] provided the recurrent version of HNNs, namely the symplectic recurrent neural networks (SRNNs). In addition, it experimentally justified that the numerical integrator applied by HNN is preferred to be a symplectic one. Regarding this issue, [198] theoretically proved the necessity of symplectic integration for HNN according to the theory of the inverse modified equation. Both HNNs and SRNNs are, in fact, the indirect methods to identify the

flow of the system, by recovering the Hamiltonian H first, then performing prediction using a numerical integrator (better be symplectic) again to solve the learned system. Hence, the HNN-based models are inefficient in the prediction process as well as in the training process, due to the need to compute the gradient of $\widetilde{H}$. Another strategy is to learn the phase flow of the Hamiltonian system directly, based on the prior knowledge of the symplecticity of the Hamiltonian flow as aforementioned. [23] used the two-layer Hamiltonian network constructed by the Verlet method [84] to achieve reversibility and symplecticity. [18] designed the architecture using the proposed symplectic additive coupling layer as its activation layer, and the pre–Iwasawa decomposed symplectic matrix [45] as its symplectic linear layer. Moreover, [125] provided a symplectic transformation by employing the real NVP [52]. In recent work in [183] the authors constructed symmetric networks in Taylor expansion form to learn the gradient of the Hamiltonian, then combined them together by a fourth-order symplectic integrator to constitute a symplectic map.

All of the aforementioned symplectic-structured networks lack the theoretical guarantees for their representability, and especially some of them are indeed unable to approximate arbitrary symplectic maps. Additionally, most of them require the learned system to be a separable Hamiltonian system, defined as follows:

Definition 3. The Hamiltonian system (2.1) is separable if

$$H(p,q) = T(p) + U(q), \quad p, q \in \mathbb{R}^d.$$

In this chapter, we develop symplectic networks (SympNets) to learn the symplectic flow of the Hamiltonian system. In fact, SympNets are able to approximate arbitrary symplectic maps within the set of symplectic maps itself. To the best of our knowledge, this is the first work which can achieve this result with theoretical

guarantees. Prior knowledge is incorporated in the sense that the searching space of the neural network is greatly reduced, and the optimization can be performed more effectively. We list below several key advantages of SympNets that will be documented in detail later:

- SympNets are able to approximate arbitrary symplectic maps in the C^r norm given appropriate activation functions, such as the sigmoid, hence, they are able to learn the phase flow of arbitrary Hamiltonian systems.
- SympNets do not require the learned Hamiltonian systems to be separable.
- SympNets can learn the continuous time evolution of dynamics in extended version as stated in Section 2.5.2.3.
- SympNets can handle the data points resulting from long time steps.
- SympNets are highly efficient in training and prediction, as they behave like a standard neural network without the need of extra computation of the gradient during both training and prediction processes or the need of performing numerical integration in the prediction stage as HNN-based models do.
- SympNets show great generalization power with an incredibly small network size, as shown in the experiments of the pendulum example, reflecting the expressivity of SympNets.
- SympNets are reversible so that the values at the forward passing stage need not be stored.
- SympNets can be extended to recurrent version without any modification, compared to SRNNs.

The rest of this chapter is organized as follows. Section 2.2 briefly summarizes the main problem we aim to solve. The detailed process of constructing the SympNets is shown in Section 2.3. In Section 2.4, we present the theoretical results for SympNets. Section 2.5 presents the simulation results for several Hamiltonian systems. We proved universal approximation theorem of SympNets in Section 2.6. We introduce SympGNN in Section 2.7 to approximate a map which is both symplectic and permutation equivariant. A summary is provided in the last section.

2.2 Problem setup

We apply a neural network model to learn the phase flow of the Hamiltonian system from data. Similar to what numerical integrators do, the trained network is used to compute the phase point after time step h of the start point y_0, i.e., the input is phase point y_0 while the output is the phase point $y_1 = \phi_h(y_0)$.

Assume that the phase flows of (2.1) are constrained in a compact space W. We first choose some phase points from W, denoted by $\{x_i\}_1^N$, and then obtain the value of time-h flow $\{y_i = \phi_h(x_i)\}_1^N$ by a high-order symplectic integrator [84]. Naturally,

$$\mathcal{T} = \{(x_i, y_i)\}_1^N$$

is viewed as the training set for learning. The neural network Φ_h as numerical integrator can be learned by minimizing the mean-squared-error loss

$$MSE = \frac{1}{2d \cdot N} \sum_{i=1}^{N} \|\Phi_h(x_i) - y_i\|^2.$$

If no prior is placed on Φ_h, it may not possess the property of symplecticity as

an integrator, which means that the Hamiltonian may not be conserved in a long-time integration. In other words, we should carefully design Φ_h to make sure it is intrinsically symplectic, if we want to make accurate long term prediction based on the learned model. The architecture of Φ_h will be shown in the next section.

2.3 Architecture

Our architecture design philosophy is based on the fact that the composition of symplectic transformations is again symplectic. In order to construct the destination symplectic map, we make an effort to search for simple linear/nonlinear symplectic maps as the building blocks of the network. We note that the building blocks should be easily parameterized so that they can be efficiently trained. An illustration of the proposed architecture is presented in Fig. 2.1.

For convenience, we employ notations used often for matrices and matrix-like maps. In this chapter, $(\cdot)$ denotes a matrix, such as

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \in \mathbb{R}^{2d\times 2d},$$

representing the $2d \times 2d$-blocked matrix with $A_1, A_2, A_3, A_4 \in \mathbb{R}^{d\times d}$, while $[\cdot]$ denotes a matrix-like map, such as

$$\begin{bmatrix} f_1 & f_2 \\ f_3 & f_4 \end{bmatrix} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}, \quad \begin{bmatrix} f_1 & f_2 \\ f_3 & f_4 \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix} := \begin{pmatrix} f_1(p) + f_2(q) \\ f_3(p) + f_4(q) \end{pmatrix},$$

representing the $2d \times 2d$-blocked matrix-like map with $f_i : \mathbb{R}^d \to \mathbb{R}^d$. Sometimes

by an abuse of notation, we also represent by the matrix $A \in \mathbb{R}^{d\times d}$ the linear map $p \to Ap$ for $p \in \mathbb{R}^d$. Hence, the identity matrix and the zero matrix $I, 0$ may represent the identity map and the zero map, respectively, when they are used in $[\cdot]$.

One of the simplest family of symplectic map from $\mathbb{R}^{2d}$ to $\mathbb{R}^{2d}$ using notations defined above is

$$f_{up}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & \nabla V \\ 0 & I \end{bmatrix}\begin{pmatrix} p \\ q \end{pmatrix}, \quad f_{low}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \nabla V & I \end{bmatrix}\begin{pmatrix} p \\ q \end{pmatrix}, \tag{2.3}$$

where $V : \mathbb{R}^d \to \mathbb{R}$ is an arbitrary function with at least C^1 regularity, and $\nabla V : \mathbb{R}^d \to \mathbb{R}^d$ is the gradient of V defined by

$$\nabla V(x) = \left(\frac{\partial V}{\partial x_1}, \frac{\partial V}{\partial x_2}, \dots \frac{\partial V}{\partial x_d}\right)^\top.$$

In fact, the composition of several f_{up} and f_{low} can approximate any symplectic map, according to 2.6. Hence one may directly model V as a neural network to obtain a "symplectic network". Nevertheless, this approach requires the computation of the gradient of a network and immediately degenerates to a Hamiltonian neural network discretized by a specific symplectic integrator. SympNets are designed to get rid of the step of calculating the gradients.

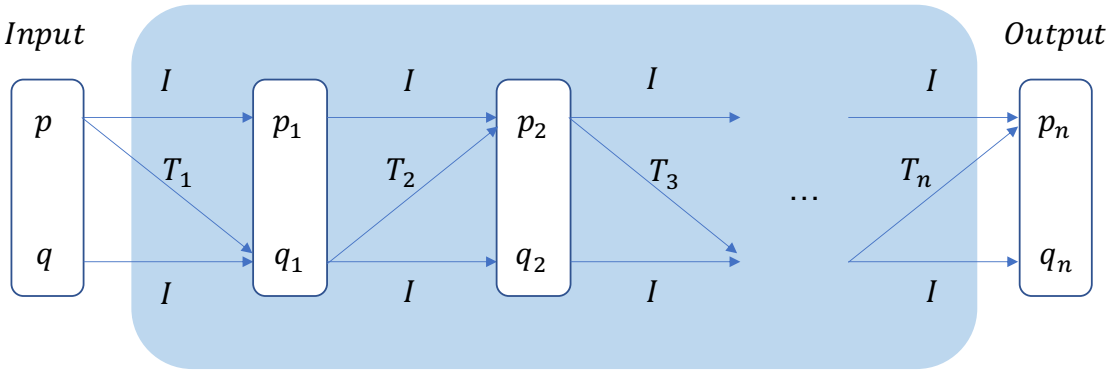


Figure 2.1: **Architecture of the SympNets.** The SympNets can be seen as a neural network with the unit triangular connection pattern, which guarantees symplecticity. Here, T_i can be chosen as S, $\tilde{\sigma}$ or $\hat{\sigma}$ (defined in Section 2.3), depending on which type of module it belongs to. Two main types of SympNets, namely LA-SympNets and G-SympNets, are considered in this chapter. For LA-SympNets, T_i are chosen to be S or $\tilde{\sigma}$ following a specific order, while for G-SympNets all the T_i are chosen to be $\hat{\sigma}$.

2.3.1 Linear modules

In reference to the linear modules, let

$$\ell_{up}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & S \\ 0 & I \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix} + b, \quad \ell_{low}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix} + b, \tag{2.4}$$
$$b \in \mathbb{R}^{2d}, \quad p, q \in \mathbb{R}^d,$$

where $S \in \mathbb{R}^{d\times d}$ is symmetric. Obviously, ℓ_{up} and ℓ_{low} are linear and symplectic, however, they are too simple to express a general linear symplectic map. In order to strengthen the expressivity of a linear layer, we compound several ℓ_{up} and ℓ_{low} alternately as

$$\mathcal{L}_n^{up}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & 0/S_n \\ S_n/0 & I \end{pmatrix}\cdots\begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix}\begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix} + b,$$

$$\mathcal{L}_n^{low}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & S_n/0 \\ 0/S_n & I \end{pmatrix} \cdots \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ S_1 & I \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + b.$$

$\mathcal{L}_n^{up}$ and $\mathcal{L}_n^{low}$ are referred to as the **linear modules** in the symplectic network. We would like the linear modules to play a similar role as the linear layers do in a fully-connected neural network. Now a problem is raised naturally, that is, are maps like $\mathcal{L}_n$ powerful enough to represent any linear symplectic map? The answer is *yes*, and we will present details in Section 2.4. It is noteworthy that during the prediction process, one may merge the triangular blocks into one matrix in advance for $\mathcal{L}_n$ to make predictions faster. In the following, we will denote the set of the linear modules as:

$$\mathcal{M}_L = \{\psi | \psi \ is \ a \ linear \ module\}.$$

Another issue worth mentioning is parameterization. Since most optimization methods in deep learning focus on unconstrained problems, it is necessary to find a representation for these modules which can be freely parameterized. In fact, the unit triangular symplectic matrices

$$\begin{pmatrix} I & S \\ 0 & I \end{pmatrix}, \quad \begin{pmatrix} I & 0 \\ S & I \end{pmatrix} \quad (S^T = S)$$

can be parameterized as

$$\begin{pmatrix} I & A + A^T \\ 0 & I \end{pmatrix}, \quad \begin{pmatrix} I & 0 \\ A + A^T & I \end{pmatrix}$$

where A is a square matrix without any constraint.

2.3.2 Activation modules

We aim to build a simple nonlinear symplectic module, which plays a similar role as the activation layer in a standard fully-connected neural network. The module is designed as

$$\begin{pmatrix} P \\ Q \end{pmatrix} = \Phi \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ diag(a)\sigma(p) + q \end{pmatrix}, \quad p, q \in \mathbb{R}^d,$$

where $a \in \mathbb{R}^d$ is the parameter to learn, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function acting element-wise as $\sigma(p) = (\sigma(p_1), \cdots, \sigma(p_d))^T$ by a slight abuse of notation. One may readily check that this map is symplectic because it can be written in the form of (2.3) with $V(p) = a^\top \cdot (\int \sigma)(p)$, where $\int \sigma$ is the antiderivative of σ. For convenience, we denote this map by

$$\mathcal{N} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \tilde{\sigma}_a & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix} := \begin{pmatrix} p \\ diag(a)\sigma(p) + q \end{pmatrix}.$$

Similar to (2.4), we specifically define

$$\mathcal{N}_{up} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & \tilde{\sigma}_a \\ 0 & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}, \quad \mathcal{N}_{low} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \tilde{\sigma}_a & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}.$$

$\mathcal{N}_{up}$ and $\mathcal{N}_{low}$ are referred to as the **activation modules** of a symplectic network. This layer plays the same role as activation layer in a standard fully-connected neural network. The universal approximation theorem of neural networks states that any continuous function can be approximated by the composition of linear units and activation units under certain constraints. Similarly, it will be shown in section 2.4 that any symplectic map can be approximated by a composition of linear modules

and activation modules. In the following, we will denote the set of the activation modules as:

$$\mathcal{M}_A = \{\psi | \psi \text{ is an activation module}\}.$$

2.3.3 Gradient modules

In addition to the modules provided in section 2.3.1 and 2.3.2, we offer an alternative choice, called the gradient module. This module will not change the approximation properties of the network, however, it offers an option, which may converge faster and result in lower testing error in some cases.

Let us define a symplectic map given an activation function σ in the following way:

$$\mathcal{G}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \hat{\sigma}_{K,a,b} & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix} := \begin{pmatrix} p \\ K^\top diag(a)\sigma(Kp+b) + q \end{pmatrix},$$

where $b \in \mathbb{R}^n$, $K \in \mathbb{R}^{n\times d}$, $a \in \mathbb{R}^n$, and n is a positive integer regarded as the width of the module. In practice, we let $n > d$ to increase the expressivity of the module. To see that $\mathcal{G}$ is indeed symplectic, one only needs to set $V(p) = a^\top \cdot (\int \sigma)(Kp+b)$ in (2.3). Now we define

$$\mathcal{G}_{up}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & \hat{\sigma}_{K,a,b} \\ 0 & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}, \quad \mathcal{G}_{low}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \hat{\sigma}_{K,a,b} & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}.$$

$\mathcal{G}_{up}$ and $\mathcal{G}_{low}$ are referred to as the **gradient modules** of the symplectic network. The "gradient module" is named following the principle that $\hat{\sigma}_{K,a,b}$ can approximate an arbitrary ∇V as shown in 2.6. These modules are inspired by the two-layer Hamiltonian network in [23], the symmetric layer in [171] and the symplectic additive

coupling layer in [18]. In the following, we will denote the set of gradient modules as:

$$\mathcal{M}_G = \{\psi | \psi \ is \ a \ gradient \ module\}.$$

2.3.4 SympNets

The **symplectic networks (SympNets)** can be informally defined as the composition of linear, activation and gradient modules. More formally, we have the following definition:

Definition 4. Consider $\{v_i\}_1^k \subset \mathcal{M}_L \cup \mathcal{M}_A \cup \mathcal{M}_G$, where $\mathcal{M}_L$, $\mathcal{M}_A$ and $\mathcal{M}_G$ are the set of linear, activation and gradient modules respectively. Let

$$\psi = v_k \circ v_{k-1} \circ \cdots \circ v_1.$$

Any such ψ is called **symplectic network (SympNet)**. Furthermore, we define the collection of SympNets as

$$\Psi = \{\psi | \psi \ is \ a \ SympNet\}.$$

Theoretically, the SympNets enjoy great algebraic and approximation properties, which will be discussed in the next section. Practically, a SympNet is highly flexible in the sense that different modules can be assembled in many different ways. The users could apply a neural architecture search (NAS) algorithm to find out the best way to assemble these modules. Here, we introduce two easily realizable ways of formulating a symplectic network, for the purpose of both proving theorems and performing numerical simulations.

Definition 5. Consider $\{v_i\}_1^{k+1} \subset \mathcal{M}_L$, $\{w_i\}_1^k \subset \mathcal{M}_A$. Let

$$\psi = v_{k+1} \circ w_k \circ v_k \circ \cdots \circ w_1 \circ v_1,$$

where ψ is called **LA-SympNet**. We define the collection of LA-SympNets as

$$\Psi_{LA} = \{\psi | \psi \ is \ a \ LA\text{-}SympNet\}.$$

Definition 6. Consider $\{u_i\}_1^k \subset \mathcal{M}_G$. Let

$$\psi = u_k \circ u_{k-1} \circ \cdots \circ u_1,$$

where ψ is called **G-SympNet**. We define the collection of G-SympNets as

$$\Psi_G = \{\psi | \psi \ is \ a \ G\text{-}SympNet\}.$$

Note that both $\Psi_{LA}, \Psi_G \subset \Psi$. Ψ_{LA} can be considered as the alternated composition of linear and activation modules while Ψ_G can be thought of as the simple combination of gradient modules. We will show that both Ψ_{LA} and Ψ_G are dense in the set of all the symplectic maps given an appropriate activation function in section 2.4.

2.4 Theory of SympNets

2.4.1 Algebraic properties

Theorem 1 (Algebraic structure)**.** The collection of all the SympNets Ψ is a group in the sense of map composition.

Proof. We know that the identity map $I \in \mathcal{M}_L \subset \Psi$ is the identity element of Ψ (group "multiplication" is given by map composition). Moreover, the associative law and the closure obviously hold by the definition of Ψ. What we need to confirm is that there exists an inverse element for any ψ, i.e., $\psi^{-1} \in \Psi$. Observe that

$$\begin{bmatrix} I & f \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -f \\ 0 & I \end{bmatrix}, \quad \begin{bmatrix} I & 0 \\ f & I \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -f & I \end{bmatrix},$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$. By substituting $f = S$, $\tilde{\sigma}_a$ and $\hat{\sigma}_{K,a,b}$ respectively, we derive that for any $\mathcal{L} \in \mathcal{M}_L,\ \mathcal{N} \in \mathcal{M}_A,\ \mathcal{G} \in \mathcal{M}_G$, it holds that

$$\mathcal{L}^{-1} \in \mathcal{M}_L \subset \Psi,\ \mathcal{N}^{-1} \in \mathcal{M}_A \subset \Psi,\ \mathcal{G}^{-1} \in \mathcal{M}_G \subset \Psi.$$

Now we consider an arbitrary SympNet $\psi = v_k \circ v_{k-1} \circ \cdots \circ v_1 \in \Psi$. It can be seen that

$$\psi^{-1} = v_1^{-1} \circ \cdots \circ v_{k-1}^{-1} \circ v_k^{-1} \in \Psi.$$

We therefore conclude that Ψ is a group. □

Being a group endows Ψ with many practically useful properties. One direct implication of theorem 1 is the following:

Corollary 1. Any SympNet $\psi \in \Psi$ is reversible.

Reversibility means that there is an analytic inverse and the value of the neural network at each layer can be computed from the output of the entire network, i.e., once $\psi(x)$ is known, we can obtain the value of $v_i \circ \cdots \circ v_1(x)$ for each $1 \leq i \leq k$. This implies that these values are unnecessary to be stored at the forward passing stage, since they can be computed directly at the backward propagation stage, which enables a memory-efficient way of implementing the neural network. This type of neural network has applications in image classification and generative modeling [51, 52, 23, 11]. Ideas in this direction can be further explored in the future.

Example 1. Here is an example for the reverse SympNet. If

$$\begin{aligned}\begin{pmatrix} P \\ Q \end{pmatrix} =& \psi \begin{pmatrix} p \\ q \end{pmatrix} \\ =& \begin{pmatrix} I & S_3 \\ 0 & I \end{pmatrix} \begin{bmatrix} I & diag(a)\sigma \\ 0 & I \end{bmatrix} \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} \begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix},\end{aligned}$$

then

$$\begin{aligned}\begin{pmatrix} p \\ q \end{pmatrix} =& \psi^{-1} \begin{pmatrix} P \\ Q \end{pmatrix} \\ =& \begin{pmatrix} I & -S_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ -S_2 & I \end{pmatrix} \begin{bmatrix} I & -diag(a)\sigma \\ 0 & I \end{bmatrix} \begin{pmatrix} I & -S_3 \\ 0 & I \end{pmatrix} \begin{pmatrix} P \\ Q \end{pmatrix}.\end{aligned}$$

Theorem 2. The collection of LA(G)-SympNets $\Psi_{LA}(\Psi_G)$ is a group.

Proof. It is similar as the proof of Theorem 1. □

2.4.2 Approximation properties

In this section, we wil present three main theorems regarding approximation properties of the SympNets. We start by introducing a few notations, which will be used later. Denote the set of symplectic matrices as

$$SP = \{H \in \mathbb{R}^{2d\times 2d} | H^T J H = J\}.$$

Similarly, we denote the set of C^r smooth symplectic map on an open set $U \subset \mathbb{R}^{2d}$ as

$$\mathcal{SP}^r(U) = \left\{ \Phi \in C^r(U;\mathbb{R}^{2d}) \middle| \left(\frac{\partial \Phi}{\partial x}\right)^\top J \left(\frac{\partial \Phi}{\partial x}\right) = J \right\}, \quad r \geq 1.$$

Also, denote

$$L_n = \left\{ \begin{pmatrix} I & 0/S_n \\ S_n/0 & I \end{pmatrix} \cdots \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} \begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} \middle| \right.$$
$$\left. S_i \in \mathbb{R}^{d\times d}, S_i^T = S_i, i = 1, 2, \cdots, n \right\}$$

where the unit upper triangular symplectic matrices and the unit lower triangular symplectic matrices appear alternately. It is clear that $L_m \subset L_n \subset SP$ for all integers $1 \leq m \leq n$.

Theorem 3. $SP = L_9$. Thus, $\mathcal{M}_L$ consists of all the linear symplectic maps.

Proof. It is known from our previous work [99] that $SP = L_9$. □

The above theorem indicates that the linear modules can parameterize any linear

symplectic map. Moreover, the depth of each linear module need not be larger than 9. In [99], we systematically present several existing modern factorizations of the matrix symplecic group, and propose the unit triangular factorization described as Theorem 3. This factorization induces the unconstrained parametrization of the matrix symplectic group by replacing the block S_i with $A_i + A_i^T$. It enables us to make use of the symplectic matrix as a module in a deep neural network, just like what we are doing here.

Restrictions on activation functions have to be made before any type of approximation theorem of neural networks can be given. Here, we introduce some necessary notations first. Let D^α be the differential operator, where $\alpha = (\alpha_1, \cdots, \alpha_m)$ with non-negative integers α_i is an ordered set of differential indexes. As an example, for $f \in C^\infty(\mathbb{R}^m)$, we have

$$D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial {x_1}^{\alpha_1} \cdots \partial {x_m}^{\alpha_m}}, \quad |\alpha| = \alpha_1 + \cdots + \alpha_m.$$

Furthermore, we define the norm on $C^r(W; \mathbb{R}^n)$ as

$$\|f\|_{C^r(W;\mathbb{R}^n)} = \sum_{|\alpha| \le r} \max_{1 \le i \le n} \sup_{x \in W} |D^\alpha f_i(x)|,$$
$$f = (f_1, \cdots, f_n)^\top \in C^r(W; \mathbb{R}^n),$$

for a compact set $W \subset \mathbb{R}^m$.

Definition 7. Let $r \in \{0\} \cup \mathbb{N}^*$ be given. σ is r-finite if $\sigma \in C^r(\mathbb{R})$ and $0 < \int |D^r(\sigma)| d\lambda < \infty$. Here $\mathbb{N}^*$ is the set of positive integers and λ is the Lebesgue measure on $\mathbb{R}$.

One of the most commonly used activation functions, the sigmoid function, satisfies this condition for any $r \in \mathbb{N}^*$. We will formalize and show this result in lemma

1.

Definition 8. Let $m, n \in \mathbb{N}^*, r \in \{0\} \cup \mathbb{N}^*$ be given, $U \subset \mathbb{R}^m$ is an open set, $S_1 \subset C^r(U; \mathbb{R}^n)$, then we say S_2 is r-uniformly dense on compacta in S_1 if $S_2 \subset S_1$ and for any $f \in S_1$, compact $W \subset U$ and any $\epsilon > 0$, there exists $g \in S_2$ such that $\|f - g\|_{C^r(W;\mathbb{R}^n)} < \epsilon$.

With the above concepts, next we present the *universal approximation theorems* for SympNets, and provide their proofs in 2.6.

Theorem 4 (Approximation theorem for LA-SympNets)**.** For any $r \in \mathbb{N}^*$ and open $U \subset \mathbb{R}^{2d}$, the set of LA-SympNets Ψ_{LA} is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ if the activation function σ is r-finite.

Theorem 5 (Approximation theorem for G-SympNets)**.** For any $r \in \mathbb{N}^*$ and open $U \subset \mathbb{R}^{2d}$, the set of G-SympNets Ψ_G is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ if the activation function σ is r-finite.

Following Theorems 4 and 5, Ψ is also r-uniformly dense on compacta in $\mathcal{SP}^r(U)$. Moreover, since Ψ and $\mathcal{SP}^r(\mathbb{R}^{2d})$ are groups, $\Psi \subset \mathcal{SP}^r(\mathbb{R}^{2d})$, we have that Ψ is a dense subgroup of $\mathcal{SP}^r(\mathbb{R}^{2d})$.

Theorems 4 and 5 give the general criterion for a SympNet to possess the universal approximation property. It is worth mentioning that the sigmoid function satisfies the condition.

Lemma 1. The sigmoid activation, $\sigma(x) = \frac{1}{1+e^{-x}}$, is r-finite for any $r \in \mathbb{N}^*$.

Proof. $\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} > 0$, so $\int |\sigma'(x)| d\lambda = \int \sigma'(x) d\lambda = 1$. By mathematical

induction, one can show that when $n \geq 2$,

$$\sigma^{(n)}(x) = \sigma'(x)P^{(n-1)}(\sigma(x)),$$

where $P^{(n-1)}(\cdot)$ is an $(n-1)$-th order polynomial, then

$$\begin{aligned} 0 < \int |\sigma^{(r)}(x)| d\lambda &\leq \int |\sigma'(x)| d\lambda \cdot (\sup_{x \in \mathbb{R}} |P^{(r-1)}(\sigma(x))|) \\ &= \sup_{y \in [0,1]} |P^{(r-1)}(y)| \\ &< \infty. \end{aligned}$$

Therefore σ is r-finite for any $r \in \mathbb{N}^*$. □

Corollary 2. The set of sigmoid-activated LA(G)-SympNets is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ for any $r \in \mathbb{N}^*$ and open $U \subset \mathbb{R}^{2d}$.

Therefore, we will use the sigmoid activation function for all of our simulation experiments presented below.

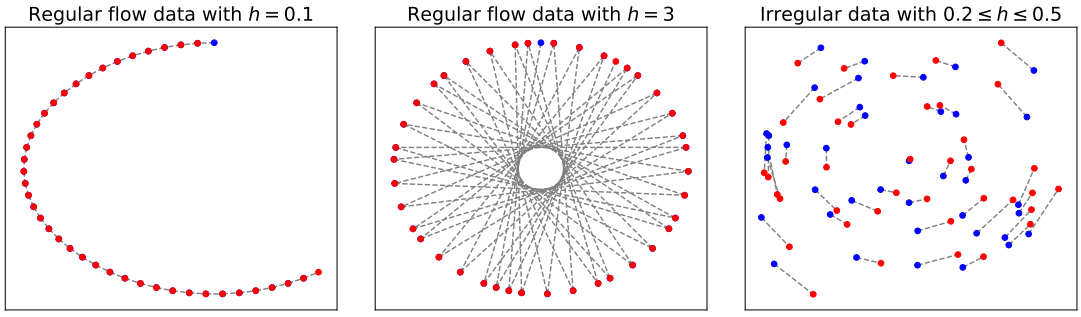


Figure 2.2: **Illustrations for datasets of pendulum.** Three types of datasets are used in the experiments of pendulum. A dash line connects a blue dot representing the initial state and a red dot representing the next state after a time step h.

2.5 Simulation results

Besides their universal approximation power, SympNets, specifically LA/G-SympNets, possess many other desirable properties in the sense that: first, they are able to generalize using limited amount of data with a small parameter space. Second, they can handle long time step prediction based models. Third, they can handle irregularly sampled data. Fourth, they can deal with non-separable Hamiltonian. Last but not least, they scale well in high dimensions. We illustrate these properties of SympNets by solving three different numerical prediction tasks, namely the pendulum, double pendulum and three-body problems.

The benchmark method used for comparison in this section is HNN [81]. The main objective to be minimized in HNN is

$$\left\| \frac{dy}{dt} - J^{-1}\nabla \widetilde{H}(y) \right\|,$$

where $y \in \mathbb{R}^{2d}$, $\widetilde{H}$ is a standard neural network. In many application scenarios, the derivative of vector fields $\frac{dy}{dt}$ is unknown, so it should be approximated using numerical discretization integrators. In fact, symplectic integrators should be applied, as is numerically justified in [32] and theoretically proved in [198]. In all of our experiments, we use the midpoint rule, a symplectic integrator of order 2, to approximate the objective:

$$\left\| \frac{x_{i+1} - x_i}{h} - J^{-1}\nabla \widetilde{H}(\frac{x_i + x_{i+1}}{2}) \right\|.$$

Once $\tilde{H}$ has been learned, we perform prediction using a 4th order symplectic integrator with finer time step to ensure the correctness of the predictions of HNNs, which indeed costs much more time compared to SympNets that make predictions directly. Since symplectic methods are applied in both training and testing proce-

dures, we will refer to the enhanced baseline model as S-HNNs, where S stands for symplectic. In this chapter, we do not require the Hamiltonians to be separable a priori for any of the test cases, so multistep or recurrent training in [32] will not be considered for S-HNNs, especially for the case of the double pendulum that is indeed non-separable.

Problem	Type	Depth	Sublayers	Width	Parameters
Pendulum: flow data	S-HNN	4	N/A	30	2K
	LA-SympNet	3	2	N/A	14
	G-SympNet	5	N/A	30	0.5K
Pendulum: irregular data	S-HNN	4	N/A	30	2K
	LA-SympNet	5	4	N/A	34
	G-SympNet	5	N/A	30	0.5K
Double pendulum	S-HNN	4	N/A	50	5K
	LA-SympNet	8	5	N/A	0.2K
	G-SympNet	8	N/A	50	2K
Three-body	S-HNN	6	N/A	50	11K
	LA-SympNet	20	4	N/A	3K
	G-SympNet	20	N/A	50	8K

Table 2.1: **Model architecture.** S-HNN uses fully-connnected neural network (FNN) as its approximator to the Hamiltonian. Depth represents the number of linear layers (linear modules) used in S-HNN (LA-SympNet), while for G-SympNet it is equal to the number of gradient modules. The number of sublayers for LA-SympNet is the number of ℓ_{up} or ℓ_{low} used to constitute each linear module. The width for G-SympNet is n, the row dimension of K in the definition of gradient module.

2.5.1 Hyper-parameters

Table 2.1 shows the architecture of the models we used for each problem. We see that LA-SympNets require a significantly smaller parameter space, especially in the case of the pendulum with $h = 0.1$, where it takes only 14 parameters to achieve the best performance. The activation function is chosen to be sigmoid for SympNets and

Problem	Type	Learning rate	Epochs
Pendulum: flow data	S-HNN	0.001	100000
	LA-SympNet	0.001	100000
	G-SympNet	0.001	100000
Pendulum: irregular data	S-HNN	0.001	100000
	LA-SympNet	0.01	100000
	G-SympNet	0.01	100000
Double pendulum	S-HNN	0.001	300000
	LA-SympNet	0.001	300000
	G-SympNet	0.001	300000
Three-body	S-HNN	0.001	300000
	LA-SympNet	0.001	300000
	G-SympNet	0.001	300000

Table 2.2: **Training parameters.** The optimizer is set to be Adam [108] for all the cases. The double pendulum and three-body problems require more epochs to converge than the pendulum problem since those problems are in higher dimensions.

Type		LA-SympNet	G-SympNet	S-HNN
$h = 0.1$	Test MSE ($\log_{10}$)	-7.3 ± 0.1	-3.0 ± 0.2	-1.6 ± 0.6
	VPT ($\log_{10}$)	3.5 ± 0.3	1.2 ± 0.1	0.3 ± 0.3
$h = 3$	Test MSE ($\log_{10}$)	-6.7 ± 0.4	-5.2 ± 0.5	N/A
	VPT ($\log_{10}$)	4.5 ± 0.4	3.7 ± 0.2	N/A
Irregular	Test MSE ($\log_{10}$)	-4.4 ± 0.4	-4.1 ± 0.5	-3.2 ± 0.2
	VPT ($\log_{10}$)	2.1 ± 0.6	1.8 ± 0.4	1.2 ± 0.1

Table 2.3: **Quantitative results for the pendulum.** The test MSE and the VPT are recorded in the form of mean $\pm$ standard deviation in log scale based on 10 independent experiments. SympNets outperform S-HNNs in all test cases.

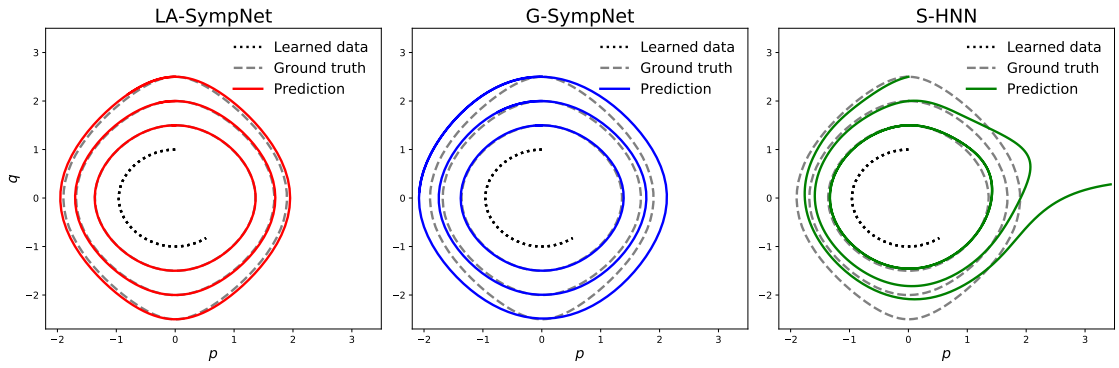


Figure 2.3: **Inferences of the outer trajectories starting at $(\mathbf{p}, \mathbf{q}) = (\mathbf{0}, \mathbf{1.5})$, $(\mathbf{0}, \mathbf{2})$, $(\mathbf{0}, \mathbf{2.5})$ for $\mathbf{h = 0.1}$.** This figure examines the extrapolation power of three different models given training data on a single trajectory starting at $(p, q) = (0, 1)$.

hyperbolic tangent (tanh) for S-HNNs. We use the normal distribution $\mathcal{N}(0, 0.01^2)$ to initialize each entry of the weight matrices in SympNets, while for S-HNNs, principal orthogonal initialization is applied. The training parameters are presented in Table 2.2.

2.5.2 The Pendulum problem

2.5.2.1 Datasets and evaluation metric

Consider the pendulum system with the Hamiltonian

$$H(p, q) = \frac{1}{2}p^2 - \cos(q).$$

In particular, we use three different datasets: (i) flow data with $h = 0.1$, (ii) flow data with $h = 3$, (iii) irregular data, to illustrate the first of three properties of SympNets described at the beginning of this section. The detailed definitions of these datasets are shown below and also in Fig. 2.2.

Flow data. The training dataset consists of $N = 40$ data points on a single trajectory starting from $x_0 = (0, 1)$ with shared time step h. These data points are grouped in pairs before being fed into the neural network, denoted as $\mathcal{T} = \{(x_{i-1}, x_i)\}_1^N$, where $x_i = \phi_h(x_{i-1}), i = 1, 2, \cdots, N$. The test dataset is given by the $k = 100$ data points following the last point in the training dataset, denoted by $X = (x_{N+1}, \cdots, x_{N+k})$. After training on $\mathcal{T}$, we use the trained network Φ_h to compute the flow starting at x_N for 100 steps, denoted by $\tilde{X} = (\tilde{x}_{N+1}, \cdots, \tilde{x}_{N+k})$. The mean squared error between X and $\tilde{X}$ is taken as the test MSE.

Irregular data. The training dataset consists of $N = 40$ grouped pairs of points randomly sampled from $[-\sqrt{2}, \sqrt{2}] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$ with time steps $\{h_i\}_1^N$ randomly chosen in $[0.2, 0.5]$, denoted as $\mathcal{T} = \{([x_i, h_i], y_i)\}_1^N$, where $y_i = \phi_{h_i}(x_i), i = 1, 2, \cdots, N$. The test dataset is generated by $k = 100$ data points on a single trajectory following $x_0 = (0, 1)$ with shared time step $h = 0.1$, denoted by $X = (x_1, \cdots, x_k)$. Same as in flow data, we generate $\tilde{X} = (\tilde{x}_1, \cdots, \tilde{x}_k)$ by the trained network and compute the mean squared error between X and $\tilde{X}$ as the test MSE. Note that we will explain how to apply the data with additional input h to SympNets later.

We compute the valid prediction time T_ϵ following a similar definition as in [188] in order to evaluate the predictive performance of different models. Suppose we are given the ground truth dataset x and prediction $\tilde{x}$. Let the normalized root mean square error be

$$\mathcal{E}(\tilde{x}) = \sqrt{\langle \frac{(x - \tilde{x})^2}{s^2} \rangle},$$

where $s \in \mathbb{R}^{2d}$ is the standard deviation in time of each state component of x, and $< \cdot >$ represents spatial average. The valid prediction time of the model is given by

$$T_\epsilon = \arg\max_{t_f} \{t_f | \mathcal{E}(\tilde{x}(t)) \leq \epsilon, \forall t \leq t_f\}.$$

In other words, T_ϵ characterizes the longest prediction window that the model remains valid. In this section, ϵ is set to 0.1.

2.5.2.2 Learning flows with fixed time steps

This problem is more difficult than the pendulum prediction problem in [81] in the sense that the training data points do not cover an entire period of the trajectory, as can be seen from Fig. 2.2. Indeed, the problem is ill-posed because there might be

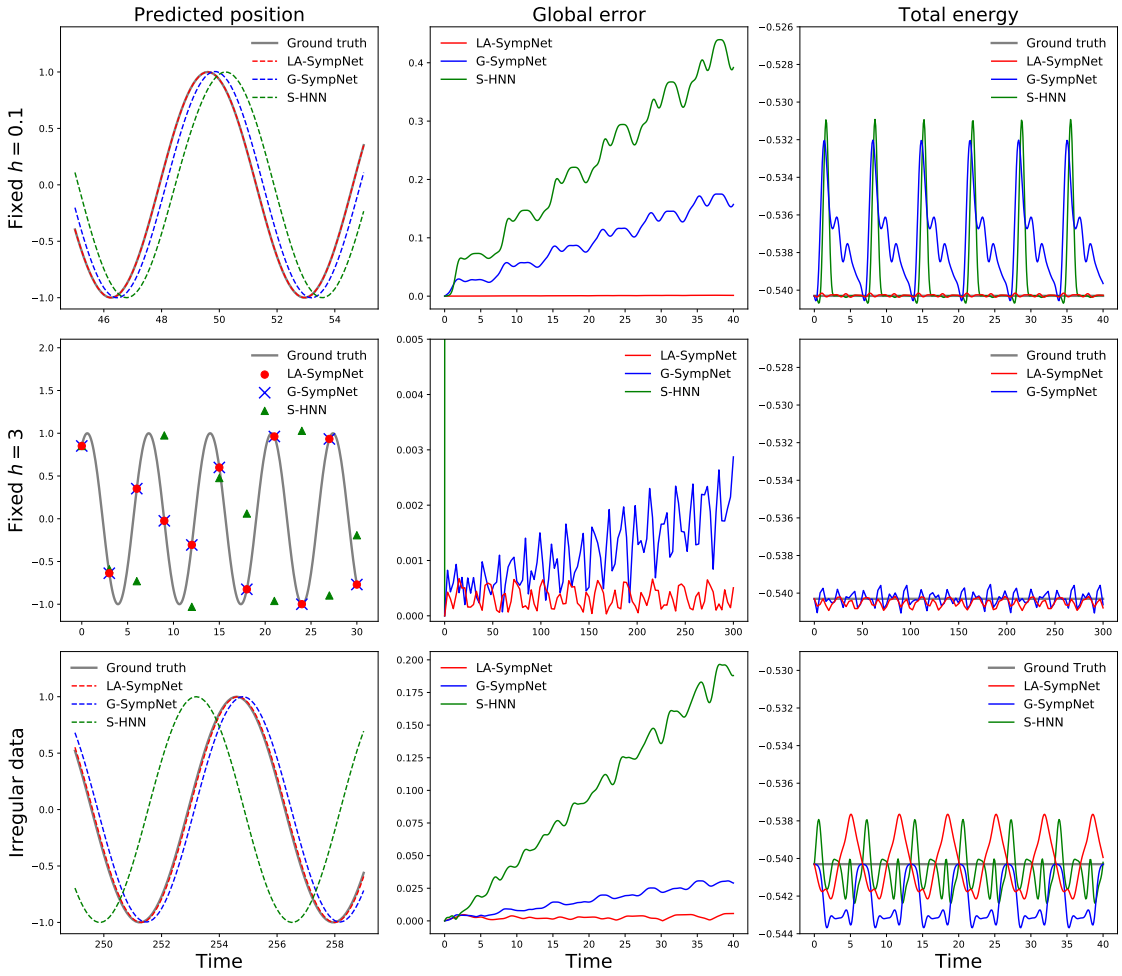


Figure 2.4: **Results of LA-SympNets, G-SympNets and S-HNNs on the three datasets for the pendulum system.** (**Left column**) The predicted positions q for the three datasets. The time windows are chosen so that the differences among the predictions made by the three methods appear. (**Middle column**) The global errors for the three datasets. It is observed that $Error_{LA} < Error_G < Error_H$ on all the datasets. (**Right column**) The total energies for the three datasets. The y-axes of the three subplots are of the same length scale so that the energy fluctuation levels can be clearly seen. The energy of S-HNN for $h = 3$ is not shown since it explodes. (**Total figure**) LA-SympNets and G-SympNets outperform S-HNNs in all cases. In fact, LA-SympNets always have the lowest global error. S-HNNs fail to learn the data with large time step due to the time discretization. LA, G and S-HNNs can all preserve the energy correctly except for S-HNN when $h = 3$.

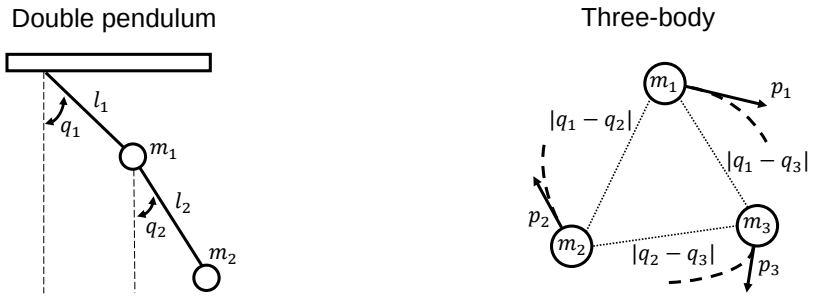


Figure 2.5: **Illustrations for the double pendulum and the three-body.** (**Left**) The double pendulum system consists of a pendulum attached directly to another one. The i-th pendulum is made of a ball of mass m_i connected to a massless rigid rod of length l_i. (**Right**) The three-body system consists of three planets of mass m_i, the motion of which is governed purely by gravitational force.

more than one Hamilton's equations whose solution could match these data points. Therefore, the learned models are expected to possess enough generalization power to learn the correct system with appropriate physical meanings.

The performance and the quantitative results of the test MSE and the VPT are shown in Fig. 2.4 and Table 2.3, respectively. Note that it does not make sense to compare their training loss, since the definitions of the loss functions for SympNets and S-HNNs are quite different. Here, 10 independent experiments are simulated for each case to obtain the means and the standard deviations. We plot the results for the best models out of 10 in the first row of Fig. 2.4. LA-SympNets, with the smallest number of parameters, achieve the lowest prediction MSE and energy fluctuation.

Fig. 2.3 shows that LA-SympNets generalize better than G-SympNets and S-HNNs on other trajectories. Given data on a single trajectory starting from $(0, 1)$, LA-SympNets can learn the correct phase flow starting from $(0, 1.5)$, $(0, 2)$, $(0, 2.5)$. It is worth mentioning that the prediction will deviate from the ground truth if the test trajectory goes farther away from the training data.

SympNets will be much easier to train when the training data is coarse-grained, or with large time steps. As mentioned before, since we do not assume the Hamiltonians are separable, one can only integrate $\tilde{H}$ by implicit symplectic schemes, which means that only one-step methods can be used to train the S-HNNs. In general, high-order implicit symplectic schemes are not compatible with the HNN-type models and could take a much longer time to train. To make the comparison fair, we still use a one-step midpoint rule here as the integrator, but one can postulate, as the time step h becomes larger, that the discretization error in S-HNNs would dominate and result in larger testing loss.

Here, we showcase a scenario when $h = 3$, which is roughly half of the period of the pendulum in our example. As shown in Table 2.3, S-HNNs fail to learn the correct dynamics of the system while SympNets continue to give the correct prediction for a long time period, according to the second row of Fig. 2.4.

2.5.2.3 Learning irregularly sampled data

Here we make an extension of SympNets to learn the data with variable time steps. As aforementioned, a symplectic module can be written like

$$v(x) = \begin{bmatrix} I & f \\ 0 & I \end{bmatrix} (x) + b,$$

where f could be S, $\tilde{\sigma}$ or $\hat{\sigma}$, depending on which type of module it belongs to, and with the bias b being zero in the cases of activation module and gradient module.

We can insert a time step h into the module as

$$v(x,h) = \begin{bmatrix} I & h \cdot f \\ 0 & I \end{bmatrix} (x) + h \cdot b.$$

Hence by extending each module in the constructed SympNets to the above form, we are able to feed the phase points x with the time steps h together as data into the extended SympNets $\psi(x,h)$ for training and testing. The results are shown in the third row of Fig. 2.4 and Table 2.3. LA, G and S-HNNs can all successfully learn the irregular data and give correct conserved energy. Specifically, we observe that SympNets perform better than S-HNNs, while LA-SympNets are slightly better than G-SympNets.

If the data used in HNN paper [81], which includes time derivatives information are given, i.e., $\mathcal{T} = \{(x_i, \dot{x}_i)\}_1^N$, then the extended SympNet $\psi(x,h)$ can also learn the fully-informed data $\mathcal{T}$ by optimizing the loss

$$MSE = \frac{1}{2d \cdot N} \sum_{i=1}^{N} \| \frac{\partial \psi}{\partial h}(x_i, 0) - \dot{x}_i \|^2.$$

This point could be further explored in the future. The capability of the extended SympNets on dealing with irregular data and fully-informed data indicates that SympNets can handle all the tasks that S-HNNs can handle, including learning the continuous time evolution of dynamics. It is certainly reasonable because the symplecticity of the phase flow encodes all the information of the dynamical system as a Hamiltonian system. Roughly speaking, $\psi(x,h)$ can be treated as a universal model representing the solution to an arbitrary Hamiltonian system.

2.5.3 The Double Pendulum problem

SympNets can readily handle the non-separable Hamiltonian systems, while S-HNNs should carefully choose the integrator if the Hamiltonian is non-separable. Here we consider a double pendulum system with the Hamiltonian

$$\begin{aligned} & H(p_1, p_2, q_1, q_2) \\ =& \frac{m_2 l_2^2 p_1^2 + (m_1 + m_2) l_1^2 p_2^2 - 2 m_2 l_1 l_2 p_1 p_2 \cos(q_1 - q_2)}{2 m_2 l_1^2 l_2^2 (m_1 + m_2 \sin^2(q_1 - q_2))} \\ & - (m_1 + m_2) g l_1 \cos q_1 - m_2 g l_2 \cos q_2. \end{aligned}$$

The double pendulum system consists of a pendulum attached directly to another one. The i-th pendulum is made of a ball of mass m_i connected to a massless rigid rod of length l_i, as is shown in Fig. 2.5. The motion of the system is driven by the local gravitational field g; q_i represents the angle of the i-th pendulum and p_i represents its corresponding canonical momentum:

$$\begin{aligned} p_1 &= (m_1 + m_2) l_1^2 \dot{q_1} + m_2 l_1 l_2 \dot{q_2} \cos(q_1 - q_2), \\ p_2 &= m_2 l_2^2 \dot{q_2} + m_2 l_1 l_2 \dot{q_1} \cos(q_1 - q_2). \end{aligned}$$

For simplicity we set $m_1 = m_2 = l_1 = l_2 = g = 1$.

Similar to the pendulum example, the training dataset is made of $N = 200$ data points on a single trajectory starting from $x_0 = (0, 0, \frac{3\pi}{7}, \frac{3\pi}{8})$ with time step $h = 0.75$. The test dataset is given by the $k = 100$ data points following the last point in the training dataset, denoted by $X = (x_{N+1}, \cdots, x_{N+k})$. The predictions made by SympNets are denoted by $\tilde{X} = (\tilde{x}_{N+1}, \cdots, \tilde{x}_{N+k})$. The mean squared error between X and $\tilde{X}$ is taken as the test loss.

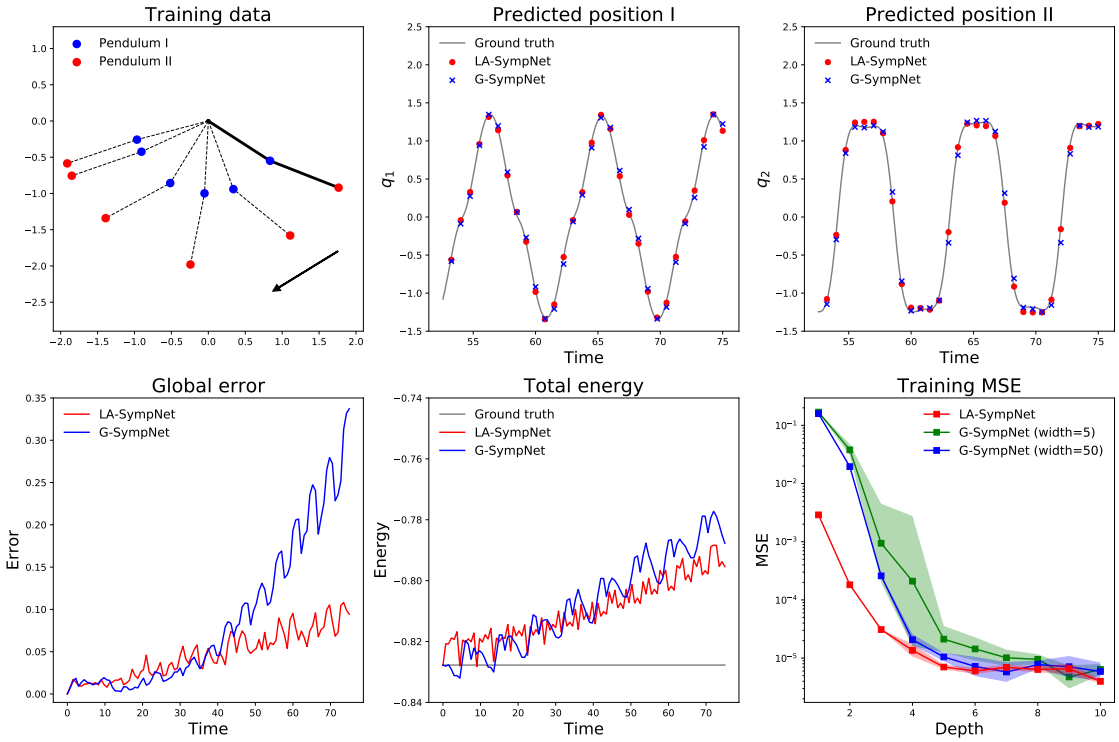


Figure 2.6: **Results for the double pendulum system.** (**Top-left**) Six consecutive points in the training dataset. The arrow represents the direction of motion. (**Top-middle and Top-right**) Predicted position q for the two pendulums, respectively. The time window is chosen so that the difference between predictions made by LA/G-SympNets appear. (**Bottom-left**) The global error versus time. LA-SympNets generalize better than G-SympNets in the long term. (**Bottom-middle**) The total energies for the predicted trajectories. (**Bottom-right**) The training MSE versus the depth. The training MSE is obtained by taking the mean of 5 independent experiments, while the shaded region represents one standard deviation.

As shown in Fig. 2.6 and Table 2.4, LA-SympNets outperform G-SympNets in the double pendulum prediction. The total energy of the trajectories predicted by SympNets matches the ground truth within a reasonable range. It is worth mentioning that S-HNNs completely fail in this task, because the time step $h = 0.75$ is so large that the discretization error in the numerical integrator dominates. This further demonstrates the advantage of SympNets when only sparsely sampled data is available. In contrast to the single pendulum case with $h = 3$, where one can remedy the S-HNN by making the educated assumption that the Hamiltonian to be learned is separable, and discretize this system with high-order symplectic schemes, here the problem is more devastating since workable high-order symplectic methods for non-separable HNNs are in general more difficult to derive, and could result in intolerable computational expense.

According to the proofs of theorems 4 and 5, the approximation power of SympNets is determined by their depth and width (for G-SympNets). Indeed the bottom-right figure of Fig. 2.6 shows that the training MSE in this experiment decreases as the network grows deeper. Lower training errors are obtained for G-SympNets of width 50 compared to that of width 5, when the depth ranges from 1 to 8. However, the difference disappears when the depth becomes sufficiently large, which indicates the fact that depth plays a more important role than width for G-SympNets. Still, a wider network is preferred since the training process could become further stabilized. Among all the three models, the standard deviation of LA-SympNets is the lowest, demonstrating that LA-SympNets are more stable compared to G-SympNets.

Problem	LA-SympNet	G-SympNet	S-HNN
Double Pendulum	-3.4 ± 0.2	-2.3 ± 0.4	N/A
Three-body	-2.2 ± 0.4	-2.9 ± 0.2	-1.8 ± 0.4

Table 2.4: **The test loss for double pendulum and three-body.** The test loss is recorded in the form of mean $\pm$ standard deviation in log scale based on 10 independent experiments.

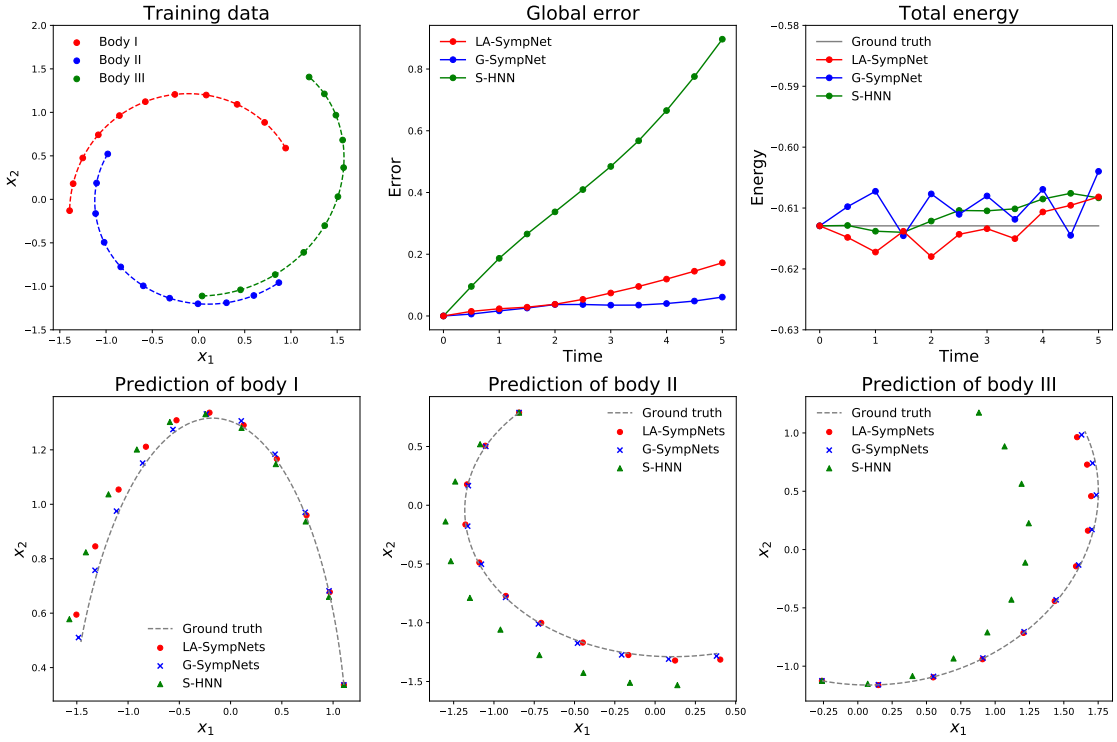


Figure 2.7: **Results for the three-body system.** (**Top-left**) One trajectory from the training dataset. The three position vectors q_1, q_2 and q_3 are plotted while the momentum vectors p_1, p_2 and p_3 are omitted for illustration purpose. (**Top-middle**) The global error versus time. The errors are calculated on one representative trajectory out of 1000. SympNets predict more accurately than S-HNNs on this and most of the other trajectories. (**Top-right**) The total energies for the predictions on the representative trajectory. (**Bottom**) Predicted position q on the representative trajectory. SympNets can make predictions which stay on the true trajectory after a relatively longer time period.

2.5.4 The Three-Body problem

To illustrate the fact that SympNets scale well to higher dimensions, we perform an experiment on the renowned three-body problem with a total number of 12 degrees of freedom. The Hamiltonian for this system is given by

$$\begin{aligned} &H(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{q_1}, \mathbf{q_2}, \mathbf{q_3}) \\ =&\frac{\mathbf{p_1}^2}{2m_1} + \frac{\mathbf{p_2}^2}{2m_2} + \frac{\mathbf{p_3}^2}{2m_3} - \frac{Gm_1m_2}{|\mathbf{q_1} - \mathbf{q_2}|} - \frac{Gm_2m_3}{|\mathbf{q_2} - \mathbf{q_3}|} - \frac{Gm_1m_3}{|\mathbf{q_1} - \mathbf{q_3}|}, \end{aligned}$$

where $\mathbf{q_i} = (q_{i1}, q_{i2})$ represents the planar coordinates of the i-th body, while $\mathbf{p_i} = (p_{i1}, p_{i2})$ and m_i are the corresponding momenta and mass, respectively; G is the gravitational constant. For simplicity, we assume $G = m_1 = m_2 = m_3 = 1$.

Due to the chaotic nature of the system, it is almost impossible for a neural network model to make correct long-term predictions as in the pendulum case. So for both training and testing, we select $k = 10$ data points with time step $h = 0.5$ on each trajectory. In total, 5000 trajectories starting at random positions are simulated, among which 4000 are used as training data while the rest serve as test data, denoted by $\{X^{(i)}\}_{i=1}^{1000}$, where $X^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_{10}^{(i)})$. Similarly, the predictions are denoted by $\{\tilde{X}^{(i)}\}_{i=1}^{1000}$. The average MSE between $X^{(i)}$ and $\tilde{X}^{(i)}$ for all $1 \leq i \leq 1000$ is taken as the test loss.

As can be seen from Table 2.4, SympNets clearly outperform S-HNNs in terms of the test loss, while G-SympNets are slightly better that LA-SympNets in this task. Still, LA-SympNets are more memory-efficient in terms of their relatively smaller parameter size. The second row of Fig. 2.7 shows that the two SympNet models are indeed comparable in their performances, while predictions made by S-HNNs completely fall off the trajectory, which is consistent with the results in [81]. All

three methods are able to conserve the total energy of the Hamiltonian system.

2.6 Proofs for universal approximation theorems

Some lemmas will be developed to prove these theorems.

Let $\Sigma_m[\sigma]$ denote the set of neural networks $f : \mathbb{R}^m \to \mathbb{R}$ with one hidden layer:

$$\Sigma_m[\sigma] = \{f(x) = a^\top \sigma(Kx + b) : \mathbb{R}^m \to \mathbb{R}| \\ a, b \in \mathbb{R}^n, K \in \mathbb{R}^{n\times m}, n \in \mathbb{N}^*\},$$

where σ is the activation function.

Lemma 2. $\Sigma_m[\sigma]$ is r-uniformly dense on compacta in $C^r(\mathbb{R}^m)$ for $m \in \mathbb{N}^*$ if σ is r-finite.

Proof. The proof can be found in [91]. □

Lemma 2 indicates that neural networks with one hidden layer can approximate a function and its derivatives simultaneously, if the function satisfies certain regularity criteria.

Lemma 3. Suppose σ is r-finite, $V \in C^{r+1}(\mathbb{R}^d)$, denote

$$f\begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & \nabla V \\ 0 & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}, \quad p, q \in \mathbb{R}^d,$$

where ∇V stands for the gradient of V, then for any compact $W \subset \mathbb{R}^{2d}$ and $\epsilon > 0$, there exists $g \in \mathcal{M}_G$ such that $\|f - g\|_{C^r(W;\mathbb{R}^{2d})} < \epsilon$.

Proof. Let $W_q = \{q \in \mathbb{R}^d | (p,q) \in W\}$. According to Lemma 2, there exists $\phi(x) = a^\top (\int \sigma)(Kx+b) \in \Sigma_d[\int \sigma]$, such that $\|V-\phi\|_{C^{r+1}(W_q;\mathbb{R})} < \epsilon$, where $\int \sigma$ is $(r+1)$-finite. It can be seen that

$$\|\nabla V - \nabla \phi\|_{C^r(W_q;\mathbb{R}^d)} \leq C\|V-\phi\|_{C^{r+1}(W_q;\mathbb{R})} < C\epsilon$$

for a positive constant C, which further implies

$$\| \begin{bmatrix} I & \nabla V \\ 0 & I \end{bmatrix} - \begin{bmatrix} I & \nabla \phi \\ 0 & I \end{bmatrix} \|_{C^r(W;\mathbb{R}^{2d})} = \|\nabla V - \nabla \phi\|_{C^r(W_q;\mathbb{R}^d)} < C\epsilon. \tag{2.5}$$

Note that $\nabla \phi(q) = K^\top diag(a)\sigma(Kq+b)$ and let

$$g \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & \nabla \phi \\ 0 & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + K^\top diag(a)\sigma(Kq+b) \\ q \end{pmatrix},$$

which is a gradient module in $\mathcal{M}_G$, then by (2.5), $\|f-g\|_{C^r(W;\mathbb{R}^{2d})} < C\epsilon$. □

By symmetry, $f^* \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} I & 0 \\ \nabla V & I \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix}$ can also be approximated by elements in $\mathcal{M}_G$ in the same way as in Lemma 3. According to [187], composition of Henon-like maps can approximate arbitrary symplectic maps. Thus, the problem reduces to approximate Henon-like maps by appropriate combination of f and f^*.

Definition 9. The symplectic maps of the following form

$$\mathcal{H}[V] \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} 0 & I \\ -I & \nabla V \end{bmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} q \\ -p + \nabla V(q) \end{pmatrix}$$

for $V \in C^1(\mathbb{R}^d)$ are called Henon-like maps.

Lemma 4. Let $U \subset \mathbb{R}^{2d}$ be an open set, then for any $F \in \mathcal{SP}^r(U)$, compact $W \subset U$, $\epsilon > 0$, there exists a sequence of $V_1, V_2, \cdots, V_n \in C^{r+1}(\mathbb{R}^d)$, such that

$$\|F - \mathcal{H}[V_n] \circ \cdots \circ \mathcal{H}[V_1]\|_{C^r(W;\mathbb{R}^{2d})} < \epsilon$$

Proof. The proof can be found in [187]. □

Proof of Theorem 5. Let $W \subset U$ be a compact set, $\epsilon > 0$. For $V \in C^{r+1}(\mathbb{R}^d)$, we have

$$\begin{bmatrix} 0 & I \\ -I & \nabla V \end{bmatrix} = \begin{bmatrix} I & 0 \\ \nabla V & I \end{bmatrix} \begin{bmatrix} I & I \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -I & I \end{bmatrix} \begin{bmatrix} I & I \\ 0 & I \end{bmatrix},$$

which shows each r-th smooth Henon-like map can be represented as the composition of elements in

$$\mathcal{F}_G = \left\{ \begin{bmatrix} I & \nabla V \\ 0 & I \end{bmatrix} \middle| V \in C^{r+1}(\mathbb{R}^d) \right\} \cup \left\{ \begin{bmatrix} I & 0 \\ \nabla V & I \end{bmatrix} \middle| V \in C^{r+1}(\mathbb{R}^d) \right\}. \tag{2.6}$$

According to Lemma 4 and the above fact, any $F \in \mathcal{SP}^r(U)$ can be approximated by a sequence of $f_1, \cdots, f_n \in \mathcal{F}_G$ as

$$\|F - f_n \circ \cdots \circ f_1\|_{C^r(W;\mathbb{R}^{2d})} < \epsilon.$$

Now we only need to prove the proposition: there exists $g_1, \cdots, g_n \in \mathcal{M}_G$ such that

$$\|f_n \circ \cdots \circ f_1 - g_n \circ \cdots \circ g_1\|_{C^r(W;\mathbb{R}^{2d})} < \epsilon.$$

Denote

$$W_1 = W, \quad W_i = T[f_{i-1}(W_{i-1})], \quad i = 2, 3, \cdots, n,$$

where the operator T is defined as

$$T[A] = \{x \in \mathbb{R}^{2d} | \exists\ y \in A\ s.t.\ \|x - y\|_\infty \leq 1\}$$

for compact $A \subset \mathbb{R}^{2d}$. It is easy to verify that W_i is compact for $i = 1, 2, 3, \cdots, n$. Let $\widetilde{W} = W_1 \cup W_2 \cup \cdots \cup W_n$. For $\epsilon_1, \cdots, \epsilon_n > 0$, there exists $g_1, \cdots, g_n \in \mathcal{M}_G$ such that

$$\|f_i - g_i\|_{C^r(\widetilde{W};\mathbb{R}^{2d})} < \min(\epsilon_i, 1), \quad i = 1, 2, \cdots, n,$$

according to Lemma 3. With the definition of g_i, we know

$$\begin{aligned} &\|g_i(x) - f_i(x)\|_\infty \\ \leq &\sup_{x \in \widetilde{W}} \|g_i(x) - f_i(x)\|_\infty \leq \|f_i - g_i\|_{C^r(\widetilde{W};\mathbb{R}^{2d})} < 1, \quad \forall x \in W_i, \end{aligned}$$

which derives that $g_i(W_i) \subset W_{i+1}$, hence we have

$$\begin{aligned} f_i \circ \cdots \circ f_1(W) \subset W_i \subset \widetilde{W}, \\ g_i \circ \cdots \circ g_1(W) \subset W_i \subset \widetilde{W}, \quad i = 1, \cdots, n. \end{aligned}$$

Let $f_i = (f_i^{(1)}, \cdots, f_i^{(2d)})^\top$. Notice that

$$D^\alpha(f_n^{(k)} \circ f_{n-1} \circ \cdots \circ f_1)(x) = P_{\alpha,n,k}([D^\beta f_i^{(j)}(f_{i-1} \circ \cdots \circ f_1(x))]_{\beta,i,j})$$

which means $D^\alpha(f_n^{(k)} \circ f_{n-1} \circ \cdots \circ f_1)(x)$ can be represented as a polynomial depending on α, n, k with respect to the parameters $D^\beta f_i^{(j)}(f_{i-1} \circ \cdots \circ f_1(x))$ for $|\beta| \leq r, 1 \leq i \leq n, 1 \leq j \leq 2d$.

As $f_i \in C^r(\mathbb{R}^{2d}; \mathbb{R}^{2d}) \subset C^1(\mathbb{R}^{2d}; \mathbb{R}^{2d})$, there holds Lipschitz condition on $\widetilde{W}$ for

each $f_j \circ \cdots \circ f_{i+1} \circ f_i$ with a shared coefficient L:

$$\begin{aligned}&\|f_j \circ \cdots \circ f_{i+1} \circ f_i(x) - f_j \circ \cdots \circ f_{i+1} \circ f_i(y)\|_\infty \\ \leq & L\|x-y\|_\infty, \quad \forall x, y \in \widetilde{W},\ \forall 1 \leq i \leq j \leq n.\end{aligned}$$

Then for any $x \in W$,

$$\begin{aligned}&\|f_i \circ \cdots \circ f_1(x) - g_i \circ \cdots \circ g_1(x)\|_\infty \\ \leq & \sum_{k=1}^{i} \|f_i \circ \cdots f_{k+1} \circ f_k \circ g_{k-1} \circ \cdots \circ g_1(x) \\ & - f_i \circ \cdots f_{k+1} \circ g_k \circ g_{k-1} \circ \cdots \circ g_1(x)\|_\infty \\ \leq & L \cdot \left(\sum_{k=1}^{i-1} \|f_k \circ g_{k-1} \circ \cdots \circ g_1(x) - g_k \circ g_{k-1} \circ \cdots \circ g_1(x)\|_\infty \right) \\ & + \|f_i \circ g_{i-1} \circ \cdots \circ g_1(x) - g_i \circ g_{i-1} \circ \cdots \circ g_1(x)\|_\infty \\ \leq & \max(L,1) \cdot \left(\sum_{k=1}^{i} \|f_k - g_k\|_{C^r(\widetilde{W};\mathbb{R}^{2d})} \right) \\ \leq & \max(L,1)(\sum_{k=1}^{i} \epsilon_k) \leq \max(L,1)(\sum_{k=1}^{n} \epsilon_k).\end{aligned}$$

Since $D^\beta f_i^{(j)}$ are uniformly continuous on $\widetilde{W}$,

$$\begin{aligned}\lim_{\epsilon_1, \cdots, \epsilon_n \to 0} \sup_{x \in W} & |D^\beta f_i^{(j)}(f_{i-1} \circ \cdots \circ f_1(x)) \\ & - D^\beta f_i^{(j)}(g_{i-1} \circ \cdots \circ g_1(x))| = 0,\end{aligned}$$

consequently

$$\begin{aligned}\lim_{\epsilon_1, \cdots, \epsilon_n \to 0} \sup_{x \in W} & |D^\beta f_i^{(j)}(f_{i-1} \circ \cdots \circ f_1(x)) \\ & - D^\beta g_i^{(j)}(g_{i-1} \circ \cdots \circ g_1(x))| = 0,\end{aligned}$$

due to

$$\begin{aligned}
&|D^\beta f_i^{(j)}(f_{i-1}\circ\cdots\circ f_1(x)) - D^\beta g_i^{(j)}(g_{i-1}\circ\cdots\circ g_1(x))| \\
\leq&|D^\beta f_i^{(j)}(f_{i-1}\circ\cdots\circ f_1(x)) - D^\beta f_i^{(j)}(g_{i-1}\circ\cdots\circ g_1(x))| \\
&+\|f_i - g_i\|_{C^r(\widetilde{W};\mathbb{R}^{2d})} \\
<&|D^\beta f_i^{(j)}(f_{i-1}\circ\cdots\circ f_1(x)) - D^\beta f_i^{(j)}(g_{i-1}\circ\cdots\circ g_1(x))| \\
&+\epsilon_i.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\lim_{\epsilon_1,\cdots,\epsilon_n\to 0}\|f_n\circ\cdots\circ f_1 - g_n\circ\cdots\circ g_1\|_{C^r(W;\mathbb{R}^{2d})} \\
=&\lim_{\epsilon_1,\cdots,\epsilon_n\to 0}\sum_{|\alpha|\leq r}\max_{1\leq k\leq n}\sup_{x\in W}|D^\alpha(f_n^{(k)}\circ f_{n-1}\circ\cdots\circ f_1)(x) \\
&-D^\alpha(g_n^{(k)}\circ g_{n-1}\circ\cdots\circ g_1)(x)| \\
=&\sum_{|\alpha|\leq r}\max_{1\leq k\leq n}\lim_{\epsilon_1,\cdots,\epsilon_n\to 0}\sup_{x\in W}|P_{\alpha,n,k}([D^\beta f_i^{(j)}(f_{i-1}\circ\cdots\circ f_1(x))]_{\beta,i,j}) \\
&-P_{\alpha,n,k}([D^\beta g_i^{(j)}(g_{i-1}\circ\cdots\circ g_1(x))]_{\beta,i,j})| \\
=&0,
\end{aligned}$$

where the last equality holds since $D^\beta f_i^{(j)}(f_{i-1}\circ\cdots\circ f_1(W))$ and $D^\beta g_i^{(j)}(g_{i-1}\circ\cdots\circ g_1(W))$ are uniformly bounded in a larger compact set for all β, i, j as $\{\epsilon_i\}\to 0$, as well as $P_{\alpha,n,k}$ is uniformly continuous on that bounded compact set. Hence, the proposition has been completed. □

Proof of Theorem 4. Recall that

$$\mathcal{M}_G = \left\{ \begin{bmatrix} I & \hat{\sigma}_{K,a,b} \\ 0 & I \end{bmatrix} \middle| K \in \mathbb{R}^{n\times d}, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\} \cup \left\{ \begin{bmatrix} I & 0 \\ \hat{\sigma}_{K,a,b} & I \end{bmatrix} \middle| K \in \mathbb{R}^{n\times d}, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\},$$

Here we rewrite $\mathcal{M}_G$ in a slightly different form as

$$\mathcal{M}_G = \left\{ \begin{bmatrix} I & \hat{\sigma}_{K,a,b} \\ 0 & I \end{bmatrix} \middle| K \in \mathbb{R}^{nd\times d}, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\} \cup \left\{ \begin{bmatrix} I & 0 \\ \hat{\sigma}_{K,a,b} & I \end{bmatrix} \middle| K \in \mathbb{R}^{nd\times d}, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\},$$

by extending K, a, b with some zero rows to meet the requirement of width being multiple of d. Furthermore, denote

$$\begin{aligned} &\mathcal{K}_n \\ =&\{K \in \mathbb{R}^{nd\times d} | K = (K_1^\top, \cdots, K_n^\top)^\top, K_i \in \mathbb{R}^{d\times d}, \det(K_i) \neq 0\}, \end{aligned}$$

$$\widetilde{\mathcal{M}}_G = \left\{ \begin{bmatrix} I & \hat{\sigma}_{K,a,b} \\ 0 & I \end{bmatrix} \middle| K \in \mathcal{K}_n, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\} \cup \left\{ \begin{bmatrix} I & 0 \\ \hat{\sigma}_{K,a,b} & I \end{bmatrix} \middle| K \in \mathcal{K}_n, a, b \in \mathbb{R}^{nd}, n \in \mathbb{N}^* \right\},$$

and

$$\widetilde{\Psi}_G = \{\psi = u_k \circ \cdots \circ u_1 | u_i \in \widetilde{\mathcal{M}}_G, k \in \mathbb{N}^*\} \subset \Psi_G.$$

Given any compact set $W \subset U$ and $\psi_{\{K_i,a_i,b_i\}} = u_{K_k,a_k,b_k} \circ \cdots \circ u_{K_1,a_1,b_1} \in \Psi_G$, we

can easily verify that

$$\lim_{\{\widetilde{K}_i\}\to\{K_i\}} \|\psi_{\{K_i,a_i,b_i\}} - \psi_{\{\widetilde{K}_i,a_i,b_i\}}\|_{C^r(W;\mathbb{R}^{2d})} = 0, \quad \psi_{\{\widetilde{K}_i,a_i,b_i\}} \in \widetilde{\Psi}_G,$$

since $\mathcal{K}_n$ is dense in $\mathbb{R}^{nd\times d}$ and

$$f(\{\widetilde{K}_i\}) = \|\psi_{\{K_i,a_i,b_i\}} - \psi_{\{\widetilde{K}_i,a_i,b_i\}}\|_{C^r(W;\mathbb{R}^{2d})}$$

is continuous with respect to $\{\widetilde{K}_i\}$. Therefore $\widetilde{\Psi}_G$ is r-uniformly dense on compacta in Ψ_G, furthermore, is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ by Theorem 5.

On the other hand, given

$$\begin{aligned}
&\phi\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ K^\top diag(a)\sigma(Kp+b)+q \end{pmatrix} \in \widetilde{\mathcal{M}}_G,\\
&K = (K_1^\top, \cdots, K_n^\top)^\top \in \mathcal{K}_n,\\
&a = (a_1^\top, \cdots, a_n^\top)^\top, a_i \in \mathbb{R}^d,\\
&b = (b_1^\top, \cdots, b_n^\top)^\top, b_i \in \mathbb{R}^d,
\end{aligned}$$

define

$$\begin{aligned}
v_i\begin{pmatrix} p \\ q \end{pmatrix} = &\begin{pmatrix} K_i^{-1} & 0 \\ 0 & K_i^T \end{pmatrix}\begin{bmatrix} I & 0 \\ diag(a_i)\sigma & I \end{bmatrix}\\
&\left(\begin{pmatrix} K_i & 0 \\ 0 & K_i^{-T} \end{pmatrix}\begin{pmatrix} p \\ q \end{pmatrix} + \begin{pmatrix} b_i \\ 0 \end{pmatrix}\right) - \begin{pmatrix} K_i^{-1}b_i \\ 0 \end{pmatrix}
\end{aligned}$$

for $i = 1, \cdots, n$. Theorem 3 points out that $v_i \in \Psi_{LA}$, and one may readily check that $\phi = v_n \circ \cdots \circ v_1$. Subsequently, we know $\widetilde{\Psi}_G \subset \Psi_{LA}$, thus Ψ_{LA} is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$. □

2.7 Symplectic graph neural networks

It was empirically observed that SympNets, along with other Hamiltonian-based neural networks including HNNs struggle to identify high-dimensional many-body systems from data if no additional structure are added to the neural network [183]. One particular structure, that could potentially is alleviate this issue is permutation equivariance.

Definition 10. A map $f : \mathbb{R}^{n\times d} \to \mathbb{R}$ is called permutation invariant if $f(P\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^{n\times d}$ for all permutation matrix P.

Definition 11. A map $\phi : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ is called permutation equivariant if $\phi(P\boldsymbol{x}) = P\phi(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^{n\times d}$ for all permutation matrix P.

Definition 12. Suppose $\boldsymbol{x} = \begin{pmatrix} x_1 & \dots & x_m \end{pmatrix} \in \mathbb{R}^{n\times m}$. Define the flatten map $fl : \mathbb{R}^{n\times m} \to \mathbb{R}^{nm}$ by $fl(\boldsymbol{x}) = \begin{pmatrix} x_1 \\ \dots \\ x_m \end{pmatrix}$. A map $\phi : \mathbb{R}^{n\times 2d} \to \mathbb{R}^{n\times 2d}$ is called symplectically permutation equivariant if (i) $fl^{-1} \circ \phi \circ fl$ is symplectic and (ii) $\phi\left(\begin{pmatrix} P\boldsymbol{p} & P\boldsymbol{q} \end{pmatrix}\right) = \begin{pmatrix} P\tilde{\boldsymbol{p}} & P\tilde{\boldsymbol{q}} \end{pmatrix}$ for all $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{n\times d}$ for all permutation matrix P , where $\begin{pmatrix} \tilde{\boldsymbol{p}} & \tilde{\boldsymbol{q}} \end{pmatrix} = \phi\left(\begin{pmatrix} \boldsymbol{p} & \boldsymbol{q} \end{pmatrix}\right)$.

In this section, we propose the symplectic graph neural networks (SympGNNs), which are symplectically permutation equivariant, as a generalization of SympNets to handle the task of high-dimensional system identification as well as node classification. In subsection 2.7.1, we introduce the motivation to design permutation equivariant SympNets, i.e., SympGNNs. In subsection 2.7.2, we introduce the architecture of two graph neural network models, LA-SympGNNs and G-SympGNNs.

In subsection 2.7.3, we present three numerical simulations to demonstrate the capability of SympGNN in handling a variety of simulation tasks.

2.7.1 Motivation

G-SympNets as splitting schemes

Consider a separable Hamiltonian system with $H(p,q) = T(p) + V(q)$:

$$\begin{cases} \dfrac{dp(s)}{dt} = -\nabla V(q(s)), \forall s \in [0,\infty) \\ \dfrac{dq(s)}{dt} = \nabla T(p(s)), \forall s \in [0,\infty) \\ p(0) = p^{(0)},\ q(0) = q^{(0)}\ . \end{cases} \tag{2.7}$$

We denote the solution to the above equation at time t by $p(t; p^{(0)}, q^{(0)})$ and $q(t; p^{(0)}, q^{(0)})$, to highlight the dependence on initial conditions. The solution operator ϕ_t is defined by

$$\phi_t \begin{pmatrix} p^{(0)} \\ q^{(0)} \end{pmatrix} := \begin{pmatrix} p(t; p^{(0)}, q^{(0)}) \\ q(t; p^{(0)}, q^{(0)}) \end{pmatrix}. \tag{2.8}$$

A general way to approximate ϕ_t would be to use the splitting scheme which takes the form

$$\varphi_t \begin{pmatrix} p^{(0)} \\ q^{(0)} \end{pmatrix} := \prod_{i=1}^{l} \left(\begin{pmatrix} I & 0 \\ k_i \nabla T_i & I \end{pmatrix} \begin{pmatrix} I & -h_i \nabla V_i \\ 0 & I \end{pmatrix} \right) \begin{pmatrix} p^{(0)} \\ q^{(0)} \end{pmatrix}, \tag{2.9}$$

where $\sum_{i=1}^{l} h_i = \sum_{i=1}^{l} k_i = t$, $\sum_{i=1}^{l} V_i = V$, $\sum_{i=1}^{l} T_i = T$. It can be seen that (2.6) matches the form of (2.9) if we replace ∇Vs by $-h_i \nabla V_i$ and $k_i \nabla T_i$. Therefore we can interpret G-SympNet as being automatically learning a Hamiltonian system

together with its splitting scheme, when the underlying Hamiltonian is *separable*.

Theorem 6. Suppose $V_i : \mathbb{R}^{n\times d} \to \mathbb{R}$ and $T_i : \mathbb{R}^{n\times d} \to \mathbb{R}$ are permutation invariant for all $i = 1, \cdots, l$, then the alternating composition of the following two parameterized functions:

$$\begin{aligned}
&\mathcal{E}_i^{low}\begin{pmatrix} \boldsymbol{p} & \boldsymbol{q} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} & \boldsymbol{q} + \nabla T_i(\boldsymbol{p}) \end{pmatrix} \quad \forall \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{n\times d}, \\
&\mathcal{E}_i^{up}\begin{pmatrix} \boldsymbol{p} & \boldsymbol{q} \end{pmatrix} = \begin{pmatrix} \boldsymbol{p} + \nabla V_i(\boldsymbol{q}) & \boldsymbol{q} \end{pmatrix} \quad \forall \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{n\times d} \\
&\varphi = \mathcal{E}_l^{up} \circ \mathcal{E}_l^{low} \cdots \mathcal{E}_1^{up} \circ \mathcal{E}_1^{low} \quad \text{or} \quad \varphi = \mathcal{E}_l^{low} \circ \mathcal{E}_l^{up} \cdots \mathcal{E}_1^{low} \circ \mathcal{E}_1^{up},
\end{aligned} \tag{2.10}$$

are symplectically permutation equivariant.

This inspires us to design T_i and V_i as neural networks with permutation invariant property, in order to respect the permutation equivariance of φ.

Remark 1. While SympNets are capable of learning a splitting scheme for separable Hamiltonian systems, they also possess the ability to learn systems with nonseparable Hamiltonians.

Remark 2. We will use the notation

$$\boldsymbol{x} = \begin{pmatrix} x_1 & \cdots & x_m \end{pmatrix} = \begin{pmatrix} x^1 \\ \cdots \\ x^n \end{pmatrix} \tag{2.11}$$

to represent the columnwise and rowwise expansion of any $\boldsymbol{x} \in \mathbb{R}^{n\times m}$.

2.7.2 Symplectic graph neural networks

Consider an undirected graph $\mathcal{G} = (\mathcal{V}, E)$ consisted of n nodes. Suppose the the node features are $\boldsymbol{x} \in \mathbb{R}^{n\times m}$, where m is the number of features in each node. The adjacency matrix of graph is given by $A \in \mathbb{R}^{n\times n}$ such that $A_{ij} = A_{ji} \forall i, j \in \{1, \cdots, n\}$, which could be either weighted or unweighted. A graph encoder is a function $\psi_{en} : \mathbb{R}^{n\times m_1} \to \mathbb{R}^{n\times 2d}$, such that

$$\psi_{en}\begin{pmatrix} x^1 \\ \cdots \\ x^n \end{pmatrix} := \begin{pmatrix} \phi_{en}(x^1) \\ \cdots \\ \phi_{en}(x^n) \end{pmatrix} = \begin{pmatrix} p^1 & q^1 \\ \cdots & \cdots \\ p^n & q^n \end{pmatrix}, \tag{2.12}$$

where $\phi_{en} : \mathbb{R}^{m_1} \to \mathbb{R}^{2d}$ can be (i) an identity map or (ii) a learnable affine map or (iii) a fully connected neural network. Similarly, we define a graph decode to be $\psi_{de} : \mathbb{R}^{n\times 2d} \to \mathbb{R}^{n\times m_{out}}$, such that

$$\psi_{de}\begin{pmatrix} z^1 \\ \cdots \\ z^n \end{pmatrix} := \begin{pmatrix} \phi_{de}(z^1) \\ \cdots \\ \phi_{de}(z^n) \end{pmatrix}, \tag{2.13}$$

for $\phi_{de} : \mathbb{R}^{2d} \to \mathbb{R}^{m_{out}}$. A symplectic graph neural network (SympGNN) is defined by $\varphi_{SG} = \psi_{de} \circ \psi_{symp} \circ \psi_{en}$, where ψ_{symp} is a symplectically permutation equivariant function, which can be constructed in the following ways:

Definition 13 (G-SympGNN)**.** Let

$$T_i^{(G)}(\boldsymbol{p}) := \sum_{j=1}^{n} \phi_v^i(p^j) \tag{2.14}$$

for any $\boldsymbol{p} = \begin{pmatrix} p^1 \\ \dots \\ p^n \end{pmatrix} \in \mathbb{R}^{n\times d}$, where $\phi_v^i : \mathbb{R}^{d+m} \to \mathbb{R}$ is the node function which can be parameterized by fully-connected neural networks. Let

$$V_i^{(G)}(\boldsymbol{q}; A) := \begin{cases} \sum_{j=1}^{n} \sum_{(j,k)\in E} \phi_e^i(q^j, q^k, A_{jk}) & \text{if } A \text{ is weighted} \\ \sum_{j=1}^{n} \sum_{(j,k)\in E} \phi_e^i(q^j, q^k) & \text{otherwise} \end{cases} \tag{2.15}$$

for any $\boldsymbol{q} = \begin{pmatrix} q^1 \\ \dots \\ q^n \end{pmatrix} \in \mathbb{R}^{n\times d}$, $\phi_e^i : \mathbb{R}^{2d(+1)} \to \mathbb{R}$ is the edge function which can be parameterized by fully-connected neural networks. The alternated composition of $\mathcal{E}_i^{up}, \mathcal{E}_i^{low}$ (see eq. 2.10) constructed with T_i, V_i defined as $T_i^{(G)}$ and $V_i^{(G)}$, $i = 1, \cdots, l$, are denoted as ψ_G. The G-SympGNN is defined by $\varphi_{GSG} := \psi_{de} \circ \psi_G \circ \psi_{en}$.

Theorem 7. $T_i^{(G)}$ and $V_i^{(G)}$ are permutation invariant for all $i \in \{1, \cdots, l\}$, so ψ_G is symplectically permutation equivariant. Therefore, φ_{GSG} is permutation equivariant.

We then introduce two types of LA-SympGNNs.

Definition 14 (LA-SympGNN)**.** Define $T_i^{(L)}, V_i^{(L)}, T_i^{(N)}, V_i^{(N)} : \mathbb{R}^{n\times d} \to \mathbb{R}$ as follows:

$$\begin{aligned} T_i^{(L)}(\boldsymbol{p}) &= fl(\boldsymbol{p})^\top (K_i \otimes \square) fl(\boldsymbol{p}), \\ V_i^{(L)}(\boldsymbol{q}) &= fl(\boldsymbol{q})^\top (S_i \otimes \square) fl(\boldsymbol{q}), \\ T_i^{(N)}(\boldsymbol{p}) &= \mathbf{1}_{1\times n} \left((\int \sigma)(\boldsymbol{p}) \right) a_i, \\ V_i^{(N)}(\boldsymbol{q}) &= \mathbf{1}_{1\times n} \left((\int \sigma)(\boldsymbol{q}) \right) b_i, \end{aligned} \tag{2.16}$$

where $\otimes$ represent the Kronecker product of two tensors. $\square \in \mathbb{R}^{n\times n}$ represents

a one-step linear graph message passing operator, which for instance can be (i) $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ or (ii) $L = D - A$. The learnable parameters are $S_i \in \mathbb{R}^{d\times d}$, $K_i \in \mathbb{R}^{d\times d}$, $a_i \in \mathbb{R}^{d\times 1}$ and $b_i \in \mathbb{R}^{d\times 1}$. The alternated composition of $\mathcal{E}_i^{up}, \mathcal{E}_i^{low}$ constructed with $T_i^{(L)}$, $V_i^{(L)}$, $T_i^{(N)}$, $V_i^{(N)}$, $i = 1, \cdots l$, are denoted as ψ_{LA}. The LA-SympGNN is defined by $\varphi_{LASG} := \psi_{de} \circ \psi_{LA} \circ \psi_{en}$.

Theorem 8. $T_i^{(L)}$, $V_i^{(L)}$, $T_i^{(N)}$ and $V_i^{(N)}$ are permutation invariant for all $i \in \{1 \cdots l\}$, so ψ_{LA} is symplectically permutation equivariant. Therefore, φ_{LASG} is permutation equivariant.

Remark 3 (Connection with GCN [109] and GRAFF [49])**.** The update rule of a *linear* Graph Convolution Network (GCN) at the k-th layer is

$$\boldsymbol{x}^{(k)} = \tilde{A}\boldsymbol{x}^{(k-1)}W_k, \tag{2.17}$$

and the update rule of a *linear* gradient-flow based graph neural network (GRAFF) at the k-th layer is

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \tilde{A}\boldsymbol{x}^{(k-1)}W_k - \boldsymbol{x}^{(k-1)}diag(\omega_k) - \beta\boldsymbol{x}^{(0)}, \tag{2.18}$$

where $W \in \mathbb{R}^{d\times d}$ is enforced to be symmetric. If we set $\beta = 0, \omega_k = 0$, then the update rules become

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \tilde{A}\boldsymbol{x}^{(k-1)}W_k, \tag{2.19}$$

which is basically *linear* GCN with residual connection. If we set $\square = \tilde{A}$ and consider only the linear layers in LA-SympGNN. Then our update rule becomes similar to the update rules above in the sense that we are mapping

$$\begin{aligned} \boldsymbol{p}^{(k)} &= \boldsymbol{p}^{(k-1)} + \tilde{A}\boldsymbol{q}^{(k-1)}S_k \\ \boldsymbol{q}^{(k)} &= \boldsymbol{q}^{(k-1)} + \tilde{A}\boldsymbol{p}^{(k)}K_k \end{aligned} \tag{2.20}$$

And we also enforce S_k, K_k to be symmetric. The difference lies in the fact that the dynamics of GRAFF follows a gradient flow while the dynamics of LA-SympGNN follows a symplectic flow.

2.7.3 Simulation results for SympGNN

We consider two different types of tasks, system identification and node classification using SympGNN. The first two numerical simulations, coupled harmonic oscillator and the Lennard Jones particle examples are system identification task. The last example on the CORA dataset is an example for node classification.

For the system identification task, we are provided with the feature vector $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q}) \in \mathbb{R}^{n\times 2d}$ directly, and the encoder / decoder will be set as identity map. The goal is to learn the dynamics from the history of $\{\boldsymbol{x}(ht)\}_{t=1}^{T}$ and then predict the evolution of $\{\boldsymbol{x}(ht)\}_{t=T+1}^{\tilde{T}}$, where h is the time gap between two consecutive observations. Specifically, in the training stage, we try to identify the "one-step-forward" solution map, i.e. $\varphi : \mathbb{R}^{n\times 2d} \to \mathbb{R}^{n\times 2d}$ such that $\varphi(\boldsymbol{x}(ht)) = \boldsymbol{x}(h(t+1))\forall t \in \{1, \cdots, T-1\}$. Then in the test stage, we apply φ iteratively to rollout the prediction $\varphi(\boldsymbol{x}(hT)), \varphi\circ\varphi(\boldsymbol{x}(hT)), \cdots, \varphi^{(\tilde{T}-T)}(\boldsymbol{x}(hT))$ to predict $\{\boldsymbol{x}(ht)\}_{t=T+1}^{\tilde{T}}$. Then the mean squared error on the rolled-out trajectory is calculated.

For the node classification task, we are provided with the feature vector $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_{train} \\ \boldsymbol{x}_{test} \end{pmatrix} \in \mathbb{R}^{(n_1+n_2)\times m}$ and the classification labels $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_{train} \\ \boldsymbol{y}_{test} \end{pmatrix} \in \{1, \cdots, c\}^{(n_1+n_2)\times 1}$, where n_1 is the number of nodes in the training set and n_2 is the number of nodes in the test set, c is the number of classes. During training stage, we try to identify a map $\varphi : \mathbb{R}^{(n_1+n_2)\times m} \to \{1, \cdots, c\}^{(n_1+n_2)\times 1}$ such that

$[\varphi(\boldsymbol{x})]_{1:n_1} \approx \boldsymbol{y}_{train}$. Here $[\varphi(\boldsymbol{x})]_{i:j}$ represents the i-th to j-th rows of $\varphi(\boldsymbol{x})$. Then the goal in the test stage is to use $[\varphi(\boldsymbol{x})]_{n_1+1:n_1+n_2}$ to predict $\boldsymbol{y}_{test}$ and check the classification accuracy.

2.7.3.1 Coupled harmonic oscillator

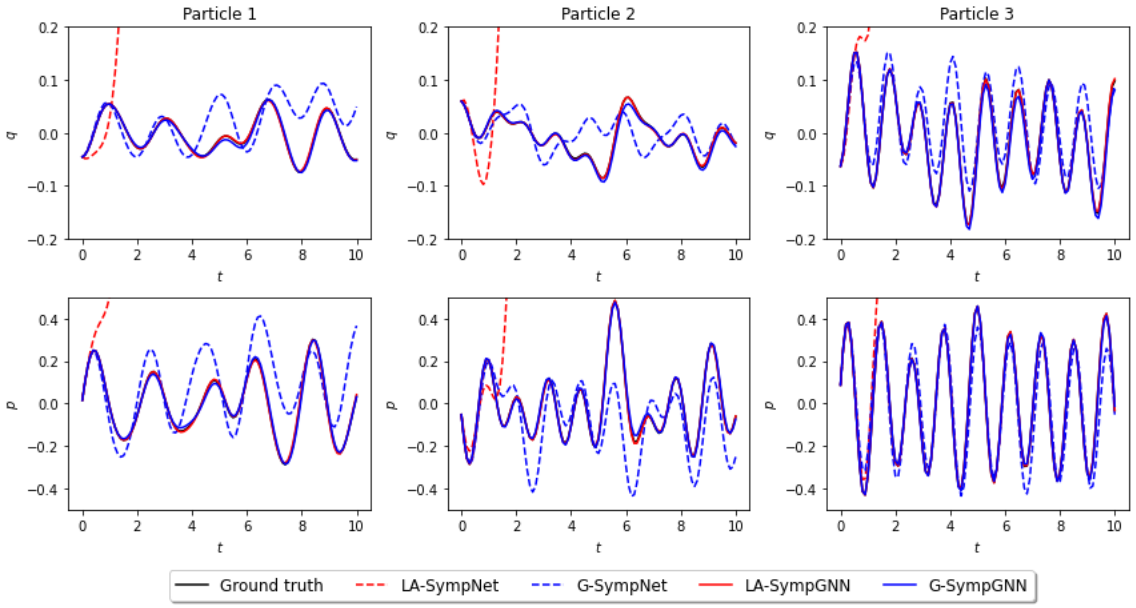


Figure 2.8: **Comparison of SympNet vs SympGNN when the training set size $T = 500$.** We randomly sample three particles from the chain and plot the trajectory in the test window. SympGNN outperforms SympNet in matching the ground truth trajectory when provided limited training data. This is because more inductive biases (permutation equivariance) is embedded.

We consider a non-dimensionalized chain with $n = 40$ particles connected by 41 springs with free boundary. All the particles share the same mass $m = 1$. The spring coefficient of the i-th spring, denoted as k_i, is sampled from $\mathcal{N}(5, 1.25^2)$, truncated below from 1. The Hamitonian of the system is given by

$$H(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2}\left(\sum_{i=1}^{n} p_i^2 + \sum_{i=1}^{n-1} k_i(q_i - q_{i+1})^2\right).$$

We further assume that all the particles are at rest at $t = 0$. The initial position

of particles are generated from $U(-2.5, 2.5)$. We set the time gap between two observations $h = 0.1$. The number of test data points is fixed $\tilde{T} - T = 100$. We further vary T to check how the prediction MSE changes with the training sample size.

In this example, the weighted adjacency matrix A is set to satisfy

$$A_{ij} = \begin{cases} 0 & \text{if } |i-j| > 1 \text{ or } \ i = j, \\ k_j & \text{if } i - j = 1, \\ k_i & \text{if } j - i = 1. \end{cases} \tag{2.21}$$

We use the graph Laplacian $\Box = L = D - A$ in the implementation of LA-SympGNN.

We run the simulation with 300000 iterations for LA-SympNet, G-SympNet, LA-SympGNN and G-SympGNN, first with the number of training samples $T = 500$. From Fig. 2.8, we can see that SympGNN outperforms SympNet in matching the ground truth trajectory when provided limited training data. This is because more inductive biases (permutation equivariance) is embedded. In Fig 2.9, we plot the total energy and MSE of the prediction when $T = 500$ and the prediction MSE as a function of training sample size T. We can see that SympGNNs both conserves the energy better and produces lower MSE compared to SympNets when $T = 500$. However, when T becomes larger, SympNets become comparable, or even better (G-SympNet) than SympGNNs, because they have more expressive power.

2.7.3.2 2d Lennard-Jones particles

We consider $n = 2000$ Argon particles governed by the Lennard-Jones potential. The data is generated with the NVE ensemble so no thermostat is involved. The training

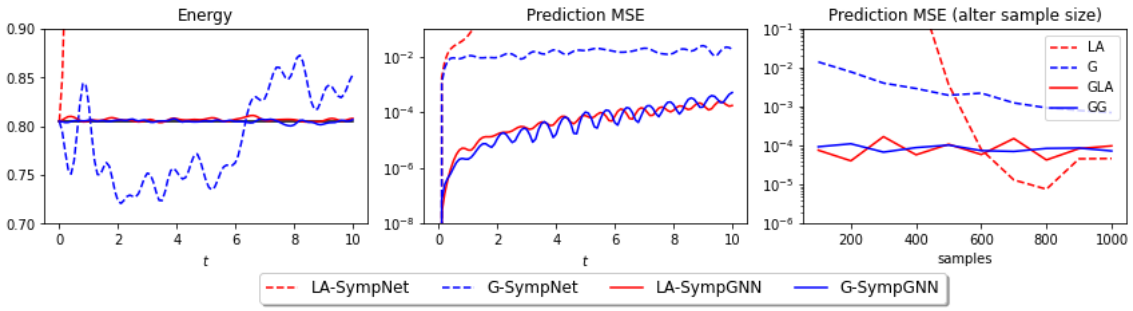


Figure 2.9: **(Left and middle) Total energy and MSE of the prediction when $T = 500$. (Right) Prediction MSE as a function of training sample size T.** We can see that SympGNNs both conserves the energy better and produces lower MSE compared to SympNets when $T = 500$. However, when T becomes larger, SympNets become comparable, or even better (G-SympNet) than SympGNNs, because they have more expressive power.

and testing data are obtained by solving the Hamiltonian system with H given by

$$H(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2}\sum_{i=1}^{n} \frac{1}{m_i}||p_i||^2 + \sum_{i=2}^{n-1} \sum_{(i,j)\in E} V\Big(||q_i - q_j||\Big).$$

We construct a graph $\mathcal{G} = (\mathcal{V}, E)$ with n nodes representing n particles. An edge is constructed when the distance between two particles become less than a threshold r_c, i.e.

$$A_{ij} = \begin{cases} 0 & \text{if } ||q_i - q_j|| > r_c, \text{ or } i = j \\ 1 & \text{otherwise .} \end{cases} \tag{2.22}$$

In molecular dynamics the Lennard-Jones potential is given by

$$V(r) = 4\epsilon\left((\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}\right) \tag{2.23}$$

, where ϵ, σ are constants representing the dispersion energy and the distance where the potential equals 0 respectively. Here we set the cutoff radius $r_c = 1.2$ nm. The simulation is conducted in a 2d periodic box with side length set to be 20 nm. We use the Composite Stormer Verlet integrator (4th order) with $h = 0.01$ ps to generate the trajectory, then sample every 10 steps as our data for training and

prediction. The training set consists of the data at first 10 ps and our goal is to predict the evolution of the system in the 10 ps following the training time window. We compared the performance of G-SympGNN with message passing neural network

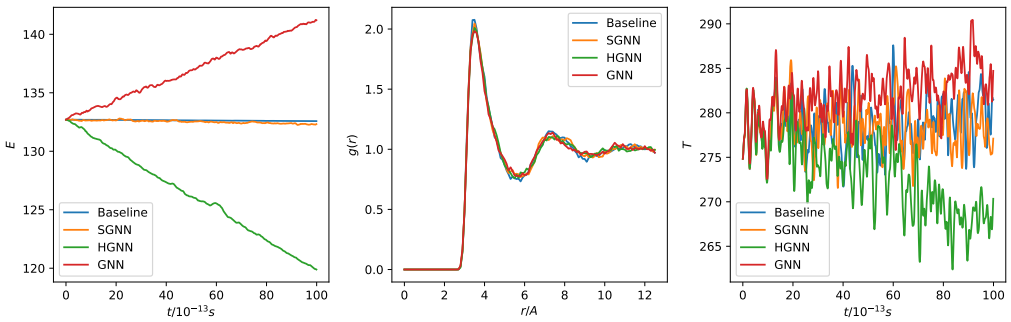


Figure 2.10: **Statistical quantities of the predicted trajectories by MPNN, HGNN and G-SympGNN.** It can be seen that G-SympGNN conserves the energy better than MPNN and HGNN.

(MPNN, [78]) and Hamiltonian graph neural network (HGNN, [172]). It can be seen that G-SympGNN conserves the energy better than MPNN and HGNN as shown in Fig 2.10.

2.7.3.3 Node classification on CORA dataset

In this numerical example, we test the performance of LA-SympGNN on CORA [136] as well as synthetic CORA [201] dataset. The CORA dataset consists of 2,708 scientific publications, each represented in a node, and classified into $c = 7$ categories. These publications are represented as nodes in a citation graph, where an edge between two nodes signifies that one publication cited the other. The feature vector $\boldsymbol{x} \in \mathbb{R}^{2708\times 1433}$ includes the content information of each publication, typically abstracts represented as bag-of-words feature vectors, and the citation relationships among them. The synthetic CORA dataset share the same features as CORA dataset, but the labels are manually generated by a modified perferential

process for a variety of heterophily level. The edge homophily ratio is defined by

$$\mathcal{H} = \frac{|\{(u,v) : (u,v) \in E \wedge \boldsymbol{y}_u = \boldsymbol{y}_v\}|}{|E|}, \tag{2.24}$$

which represents the fraction of edges which connect two nodes that share the same label.

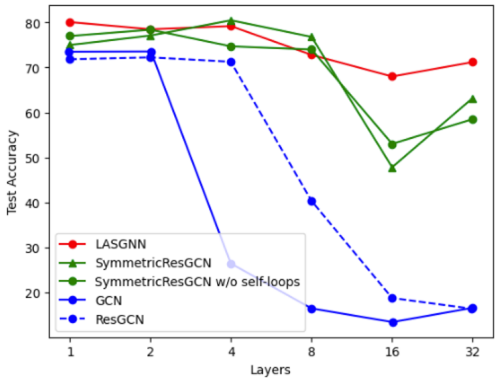
One limitation of many graph neural networks including GCN is that their test accuracies are low on low-homophily or heterophilic datasets [201]. Another related issue is that the performance of many graph neural network architecture tend to deteriorate when the number of layers grows deeper, which can be explained by the oversmoothing phenomena, i.e., the feature embeddings of graph networks become almost identical after the mapping of many layers [21].

We test the performance of LA-SympGNN versus different kinds of GCN architectures which are designed specifically for the purpose of alleviating oversmoothing phenomena:

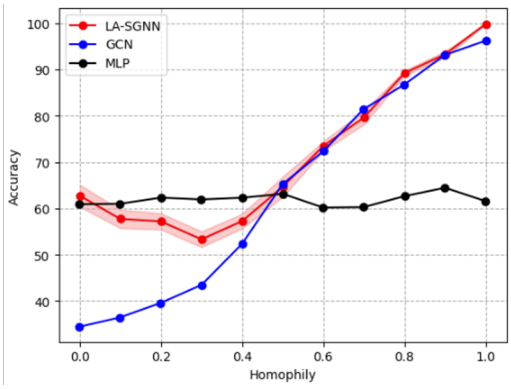
- GCN: $\boldsymbol{x}^{(k)} = \sigma(\bar{A}\boldsymbol{x}^{(k-1)}W_k)$, where $\bar{A} = D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$.
- GCN with residual: $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \sigma(\bar{A}\boldsymbol{x}^{(k-1)}W_k)$,
- symmetric GCN with residual: $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \sigma(\bar{A}\boldsymbol{x}^{(k-1)}(W_k + W_k^\top))$,
- symmetric GCN with residual and no self-loop: $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \sigma(\tilde{A}\boldsymbol{x}^{(k-1)}(W_k + W_k^\top))$

We set $\square = \tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ in LA-SympGNN. We set the encoder and decoder ψ_{en} and ψ_{de} to be an affine function (linear layer). The dimension of feature embedding

is set to be $d = 32$. It can be seen from the left side of Fig. 2.11 that the LA-SympGNN is less likely to oversmooth compared with 4 GCN benchmarks. Only a slight drop in accuracy is observed for LA-SympGNN with 32 layers of lower/upper linear/activation modules (128 layers in total). On the right hand side, we plot the prediction accuracy of LA-SympGNN at different homophily levels. It is observed that LA-SympGNN achieves GCN-like performance on the high-homophily regime, and MLP-like performance on the low-homophily regime.



(a) The prediction accuracy of LA-SympGNN and GCN w.r.t. number of layers, on the CORA dataset.



(b) The prediction accuracy of LA-SympGNN, GCN and MLP w.r.t. homophily level, on the synthetic CORA dataset.

Figure 2.11: **LA-SympGNN for node classification. (Experiment done by Alan John Varghese)** From the left figure, it can be seen that LA-SympGNN is less likely to oversmooth when the number of layers grows larger, compared to GCN. On the right figure, it can be seen that LA-SympGNN achieves GCN-like performance on the high-homophily regime, and MLP-like performance on the low-homophily regime.

2.8 Summary

The main contribution of this chapter is to provide a unified framework to infer dynamics from an arbitrary Hamiltonian system by utilizing the symplecticity of its phase flow. Just like any symplectic matrix that can be factorized into unit trian-

gular matrices, in this chapter we showed that any symplectic map, which might be nonlinear, can be approximately factorized into unit triangular matrix-like maps in a simple form, i.e. SympNets. Furthermore, the SympNets are inherently reversible, and in fact form a group. This algebraic structure indicates the possibility of building normalizing flow models from the existing architecture. Besides its intriguing theoretical properties, SympNets also exhibit superior properties over competing baseline models, i.e., HNNs through the great performance in three numerical experiments including the pendulum, double pendulum and three-body problems. In particular, LA-SympNets generalize better than G-SympNets (pendulum, double pendulum), while G-SympNets are more expressive than LA-SympNets in more challenging scenarios (three-body problem). A new theoretical contribution is the universal approximation theorems (Theorem 4 and 5) that we proved for SympNets. Finally, we proposed two graph neural network architectures, LA-SympGNN and G-SympGNN, which resembles LA-SympNet and G-SympNet, with the permutation equivariant property. It is observed that LA-SympGNN is less likely to oversmooth compared to many other graph neural network architectures.

By constructing the SympNets, we wish our work could lead to more researches that focus on utilizing the underlying geometric structures such as the symplecticity in the data. In the future, we would like to derive generative models and control algorithms based on the SympNets. Another interesting direction will be to understand theoretically, why is SympGNNs less likely to oversmooth.

Chapter Three

Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks

3.1 Introduction

The connection between dynamical systems and neural network models has been widely studied in the literature, see, for example, [22, 57, 83, 132, 131]. In general, neural networks can be considered as discrete dynamical systems with the basic dynamics at each step being a linear transformation followed by a componentwise nonlinear (activation) function. In [29] the neural ODE is introduced as a continuous-depth model instead of specifying a discrete sequence of hidden layers. Even before the introduction of neural ODEs, a series of models with similar architectures had already been proposed to learn the hidden dynamics of a dynamical system in [163, 80, 157]. Backward error analysis of the discovery of dynamics is established and enriched in [199], where the inverse modified differential equations are introduced to understand the true dynamical system that is learned when the time derivatives are approximated by numerical schemes. The methods above do not assume the specific form of the equation a priori, however, as the physical systems usually possess intrinsic prior properties, other approaches also take into account the prior information of the systems.

For many physical systems from classical mechanics, the governing equation can be expressed in terms of Hamilton's equation. We denote the d-by-d identity matrix by I_d, and let

$$J := \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix},$$

which is an orthogonal, skew-symmetric real matrix, so that $J^{-1} = J^T = -J$. The canonical Hamiltonian system can be written as

$$\dot{y} = J^{-1}\nabla H(y), \tag{3.1}$$

where $y(t) \in \mathbb{R}^{2d}$, and $H : \mathbb{R}^{2d} \to \mathbb{R}$ is the Hamiltonian, typically representing the energy of the system. It is well known that the phase flow of a Hamiltonian system is symplectic. Based on that observation, several numerical schemes which preserve symplecticity have been proposed to solve the forward problem in [65, 84, 133]. In recent works [13, 81, 162, 172, 32, 184], the primary focus has been to solve the inverse problem, i.e., identifying Hamiltonian systems from data, using structured neural networks. For example, HNNs [81] use a neural network $\tilde{H}$ to approximate the Hamiltonian H in (3.1), then learn $\tilde{H}$ by reformulating the loss function. Based on HNNs, other models were proposed to tackle problems in generative modeling [162, 184] and continuous control [196]. Another line of approach is to learn the phase flow of the system directly, while encoding the physical prior as symplecticity in the flow. Recently, we introduced symplectic networks (SympNets) with theoretical guarantees that they can approximate arbitrary symplectic maps [102]. Other networks of this type are presented in [53, 190].

In practice, requiring a dynamical system to be Hamiltonian could be too restrictive, as the system has to be described in canonical coordinates. In [35] Lagrangian Neural Networks (LNNs) were introduced, which allow the system to be expressed in Cartesian coordinates. In [67] the models of HNNs and LNNs were generalized to Constrained Hamiltonian Neural Networks (CHNNs) and Constrained Lagrangian Neural Networks (CLNNs), enabling them to learn constrained mechanical systems written in Cartesian coordinates. In other developments, autoencoder-based HNNs (AE-HNNs) [81] and Hamiltonian Generative Networks (HGNs) [184] were proposed to learn and predict the images of mechanical systems, which can be seen as Hamiltonian systems on manifolds embedded in high-dimensional spaces. Theoretically, Hamiltonian systems on manifolds written in noncanonical coordinates are equivalent to an important class of dynamical systems, namely, the Poisson systems. To

wit, the Poisson systems take the form of

$$\dot{y} = B(y)\nabla H(y),$$

where $y \in \mathbb{R}^n$, n is not necessarily an even number; H is the Hamiltonian of the Poisson system, and the matrix-valued function $B(y)$ plays the role of J^{-1} in (3.1), which induces a general Poisson bracket as defined in Section 3.2. The Darboux-Lie theorem states that a Poisson system can be turned into a Hamiltonian system by a coordinate transformation. As a consequence, structure-preserving numerical schemes for Poisson systems are normally developed by finding the coordinate transformation manually, then applying symplectic integrators on the transformed systems, see [181, 84].

Inspired by the Darboux-Lie theorem, we propose a novel neural network architecture, the Poisson neural network (PNN), to learn the phase flow of an arbitrary Poisson system, specifically, PNNs can learn any unknown diffeomorphism of Hamiltonian systems. The coordinate transformation and the phase flow of the transformed Hamiltonian system are parameterized by structured neural networks with physical priors. In the general setting, we use invertible neural networks (INNs) [51, 52] to represent the coordinate transformation. If all the data reside on a submanifold with dimension $2d < n$, autoencoders (AEs) can be applied to approximate the coordinate transformation from the coordinates in $\mathbb{R}^n$ to its local coordinates as an alternative choice. This strategy is similar to [164], which learns dynamics in the latent space discovered through an autoencoder. Compared to LNNs, PNNs are able to work on a more general coordinate system. Moreover, INN-based PNNs are able to learn multiple trajectories of a Poisson system on the whole data space simultaneously, while AE-HNNs, HGNs, CHNNs and CLNNs are only designed to work on low-dimensional submanifolds of $\mathbb{R}^n$. Further, our work lays a solid theoretical

background for all the aforementioned models, suggesting that they are learning a Poisson system explicitly or implicitly.

The main contribution of this work is the development of a framework that learns general Poisson systems with structure-preserving models. To the best of our knowledge, this is the first work which achieves this. Most of current works focus on Hamiltonian systems or constrained Hamiltonian systems. In fact, we can easily transform Poisson systems to Hamiltonian systems via a coordinate transformation. Furthermore, constrained Hamiltonian systems can be rewritten as Poisson systems and then transformed to Hamiltonian systems. Therefore, PNNs can be used to solve a wide range of problems. It is worth noting that a good learning model for Poisson systems should be structure-preserving, i.e., can preserve the Poisson structure, otherwise it is meaningless.

Another intriguing property of PNNs is that they are not only capable of learning Poisson systems, but are able to approximate an unknotted trajectory of an arbitrary autonomous system. We present related theorems, which indicate the great expressivity of PNNs. We demonstrate through computational experiments that PNNs can be practically useful in terms of learning high-dimensional autonomous systems, long-time prediction, as well as frame interpolation. PNNs also enjoy all the advantages listed in [102] as they are implemented based on the SympNets in this work. However, PNNs as a high-level architecture can also employ other modern symplectic neural networks as well as invertible neural networks.

The rest of the chapter is organized as follows. Section 3.2 introduces some necessary notation, terminology and fundamental theorems that will be used. The learning theory for PNNs is presented in Section 3.3. Section 3.4 presents the experimental results for several Poisson systems and autonomous systems. A summary is given

in the last section. Supporting materials, including the detailed implementation of PNNs, are included in the appendix.

3.2 Preliminaries

The material required for this work is based on the mathematical background of Hamiltonian system and its non-canonical form, i.e., the Poisson system. We refer the readers to [84] for more details.

3.2.1 Hamiltonian and Poisson systems

First, we formally present the definitions of the Hamiltonian system and the Poisson system. We assume that all the functions or maps involved in this chapter are as smooth as needed.

Definition 15. The canonical Hamiltonian system takes the form

$$\dot{y} = J^{-1}\nabla H(y), \tag{3.2}$$

where $H : U \to \mathbb{R}$ is the Hamiltonian typically representing the energy of the system defined on the open set $U \subset \mathbb{R}^{2d}$.

System (3.2) can also be written in a general form by introducing the Poisson bracket.

Definition 16. Let $U \subset \mathbb{R}^n$ be an open set. The Poisson bracket $\{\cdot,\cdot\} : C^\infty(U) \times C^\infty(U) \to C^\infty(U)$ is a binary operation satisfying

(i) (anticommutativity)

$$\{F, G\} = -\{G, F\},$$

(ii) (bilinearity)

$$\{aF + bG, H\} = a\{F, H\} + b\{G, H\},$$
$$\{H, aF + bG\} = a\{H, F\} + b\{H, G\}, \quad a, b \in \mathbb{R},$$

(iii) (Leibniz's rule)

$$\{FG, H\} = \{F, H\}G + F\{G, H\},$$

(iv) (Jacobi identity)

$$\{\{F, G\}, H\} + \{\{H, F\}, G\} + \{\{G, H\}, F\} = 0,$$

for $F, G, H \in C^\infty(U)$.

Consider the bracket

$$\{F, G\} = \sum_{i=1}^{d} \left(\frac{\partial F}{\partial q_i} \frac{\partial G}{\partial p_i} - \frac{\partial F}{\partial p_i} \frac{\partial G}{\partial q_i} \right) = \nabla F(y)^\top J^{-1} \nabla G(y), \tag{3.3}$$

where $y = (p, q) = (p_1, \cdots, p_d, q_1, \cdots, q_d) \in \mathbb{R}^{2d}$. One can check that (3.3) is indeed a Poisson bracket. Then system (3.2) can be written as

$$\dot{y}_i = \{y_i, H\}, \quad i = 1, \cdots, 2d,$$

where y_i in the bracket denotes the map $y \to y_i$ for $y = (y_1, \cdots, y_{2d})$ by a slight abuse of notation. Now we extend the bracket (3.3) to a general form as

$$\begin{aligned} (\{F, G\}_B)(y) &= \sum_{i,j=1}^{n} \frac{\partial F(y)}{\partial y_i} b_{ij}(y) \frac{\partial G(y)}{\partial y_j} \\ &= \nabla F(y)^\top B(y) \nabla G(y), \end{aligned} \tag{3.4}$$

where $B(y) = (b_{ij}(y))_{n\times n}$ is a smooth matrix-valued function. Note that here we do not require n to be an even number. As many crucial properties of Hamiltonian systems rely uniquely on the conditions (i)-(iv) in Definition 16, we naturally expect the bracket (3.4) to be a Poisson bracket.

Lemma 5. The bracket defined in (3.4) is anti-commutative, bilinear and satisfies Leibniz's rule as well as the Jacobi identity if and only if

$$b_{ij}(y) = -b_{ji}(y) \ for \ all \ i, j$$

and for all i, j, k

$$\sum_{l=1}^{n} \left(\frac{\partial b_{ij}(y)}{\partial y_l} b_{lk}(y) + \frac{\partial b_{jk}(y)}{\partial y_l} b_{li}(y) + \frac{\partial b_{ki}(y)}{\partial y_l} b_{lj}(y) \right) = 0.$$

Lemma 5 provides verifiable equivalence conditions for (3.4) to become a Poisson bracket. Then, we can give the definition of Poisson system, which is actually the generalized form of the Hamiltonian system.

Definition 17. If a matrix-valued function $B(y)$ satisfies Lemma 5, then (3.4) defines a Poisson bracket, and the corresponding differential system

$$\dot{y} = B(y)\nabla H(y)$$

is a Poisson system. Here H is still called a Hamiltonian.

Up to now, the Hamiltonian system and the Poisson system have been unified as

$$\dot{y}_i = \{y_i, H\}_B, \quad i = 1, \cdots, n, \tag{3.5}$$

for B satisfying Lemma 5, and the system becomes Hamiltonian when $B = J^{-1}$.

3.2.2 Symplectic map and Poisson map

The study of the phase flows of the Hamiltonian and Poisson systems focuses on the symplectic map and the Poisson map.

Definition 18. A transformation $\Phi : U \to \mathbb{R}^{2d}$ (where U is an open set in $\mathbb{R}^{2d}$) is called a symplectic map if its Jacobian matrix satisfies

$$\left(\frac{\partial \Phi}{\partial y}\right)^T J \left(\frac{\partial \Phi}{\partial y}\right) = J.$$

Definition 19. A transformation $\Phi : U \to \mathbb{R}^n$ (where U is an open set in $\mathbb{R}^n$) is called a Poisson map with respect to the Poisson system (3.5) if its Jacobian matrix satisfies

$$\left(\frac{\partial \Phi}{\partial y}\right) B(y) \left(\frac{\partial \Phi}{\partial y}\right)^\top = B(\Phi(y)).$$

In fact, the phase flow $\phi_t^H(y)$ of the Hamiltonian system is a symplectic map, while the phase flow $\phi_t^P(y)$ of the Poisson system is a Poisson map, i.e., $\phi_t^H(y)$ and $\phi_t^P(y)$ satisfy Definition 18 and 19, respectively. Based on these facts, we naturally expect the numerical methods or learning models for Hamiltonian systems and Poisson systems to preserve the intrinsic properties that $\phi_t^H(y)$ and $\phi_t^P(y)$ possess. So far the numerical techniques for Hamiltonian systems have been well developed [65, 84, 133], however, research on the Poisson systems is ongoing due to its complexity.

3.2.3 Coordinate changes and the Darboux–Lie theorem

The main idea in studying Poisson systems is to find the connection to Hamiltonian systems, which are easier to deal with. In fact, a Poisson system expressed in arbitrary new coordinates is again a Poisson system, hence we naturally tend to simplify a given Poisson structure as much as possible by coordinate transformation.

Theorem 9 (Darboux 1882, Lie 1888)**.** Suppose that the matrix $B(y)$ defines a Poisson bracket and is of constant rank $n - q = 2d$ in a neighbourhood of $y_0 \in \mathbb{R}^n$. Then, there exist functions $P_1(y), \cdots, P_d(y)$, $Q_1(y), \cdots, Q_d(y)$, and $C_1(y), \cdots, C_q(y)$ satisfying

$$\begin{array}{lll} \{P_i, P_j\} = 0 & \{P_i, Q_j\} = -\delta_{ij} & \{P_i, C_l\} = 0 \\ \{Q_i, P_j\} = \delta_{ij} & \{Q_i, Q_j\} = 0 & \{Q_i, C_l\} = 0 \\ \{C_k, P_j\} = 0 & \{C_k, Q_j\} = 0 & \{C_k, C_l\} = 0 \end{array}$$

on a neighbourhood of y_0, where δ_{ij} equals to 1 if $i = j$ else 0. The gradients of P_i, Q_i, C_k are linearly independent, so that $y \to (P_i(y), Q_i(y), C_k(y))$ constitutes a local change of coordinates to canonical form.

Corollary 3 (Transformation to canonical form)**.** Let us denote the transformation of Theorem 9 by $z = \theta(y) = (P_i(y), Q_i(y), C_k(y))$. With this change of coordinates, the Poisson system $\dot{y} = B(y)\nabla H(y)$ becomes

$$\dot{z} = B_0 \nabla K(z) \quad with \quad B_0 = \begin{pmatrix} J^{-1} & 0 \\ 0 & 0 \end{pmatrix},$$

where $K(z) = H(y)$. Writing $z = (p, q, c)$, this system becomes

$$\dot{p} = -K_q(p, q, c), \quad \dot{q} = K_p(p, q, c), \quad \dot{c} = 0.$$

Corollary 3 reveals the connection between Poisson systems and Hamiltonian systems via coordinate changes. In a forward problem, i.e., solving the Poisson system by numerical integration, transformations are available for many well-known systems to perform structure-preserving calculations [181, 84], but there does not exist a general method to search for the new coordinates of an arbitrary Poisson system, which is still an open research issue. However, the inverse problem, i.e., learning an unknown Poisson system based on data, is an easier task.

3.3 Learning theory for Poisson systems and trajectories of autonomous systems

Assume that there is a dataset $\mathcal{T} = \{(x_i, y_i)\}_1^N$ $(x_i, y_i \in \mathbb{R}^n)$ from an unknown autonomous dynamical system, that could be a Poisson system or not, satisfying $\phi_h(x_i) = y_i$ for time step h and phase flow ϕ_t. We aim to discover the dynamics using learning models, so that we can make predictions into future or perform some other computational tasks. To describe things clearly, we first give the definition of the extended symplectic map.

Definition 20. A transformation $\Phi : U \to \mathbb{R}^n$ (where U is an open set in $\mathbb{R}^n$) is called an extended symplectic map with latent dimension $2d$ if it can be written as

$$\Phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \phi(x_1, x_2) \\ x_2 \end{pmatrix}, \quad x_1 \in \mathbb{R}^{2d}, x_2 \in \mathbb{R}^{n-2d}, 2d \leq n,$$

where ϕ is differentiable, and $\phi(\cdot, x_2)$ is a symplectic map for each fixed x_2. Note that Φ degenerates to a general symplectic map when $2d = n$.

3.3.1 Poisson neural networks

We propose a high-level network architecture, i.e., the Poisson neural networks (PNNs), to learn Poisson systems or autonomous flows based on the Darboux–Lie theorem. Theorem 9 and Corollary 3 indicate that any Poisson system in n-dimensional space can be transformed to a "piecewise" Hamiltonian system, where $2d \leq n$ is the latent dimension determined by the rank of $B(y)$. The architecture is composed of three parts: (1) a coordinate transformation, (2) an extended symplectic map, (3) the inverse of the transformation, denoted by θ, Φ and θ^{-1}, respectively. For the construction of extended symplectic neural networks we refer the readers to Appendix 3.5, which is an important part of this work. The transformations θ and θ^{-1} can be implemented using two alternative approaches.

Primary architecture. We model θ as an invertible neural network, as in [51, 52], to automatically obtain its inverse θ^{-1}. Then, we learn the data by optimizing the mean-squared-error loss

$$L(\mathcal{T}) = \frac{1}{n \cdot N} \sum_{i=1}^{N} \|\theta^{-1} \circ \Phi \circ \theta(x_i) - y_i\|^2,$$

where $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ is an extended symplectic neural network with latent dimension $2d$.

Alternative architecture. We exploit an autoencoder to parameterize θ, θ^{-1} with

two different neural networks. Then, the loss is designed as

$$\begin{aligned}
&L(\mathcal{T}) \\
=&L_s(\mathcal{T}) + \lambda \cdot L_a(\mathcal{T}) \\
=&\frac{1}{2d \cdot N} \sum_{i=1}^{N} \|\Phi \circ \theta(x_i) - \theta(y_i)\|^2 + \\
&\lambda \cdot \frac{1}{n \cdot N} \sum_{i=1}^{N} (\|\theta^{-1} \circ \theta(x_i) - x_i\|^2 + \|\theta^{-1} \circ \theta(y_i) - y_i\|^2),
\end{aligned}$$

where $\Phi : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is a symplectic neural network and λ is a hyperparameter to be tuned. Note that in this case $\theta^{-1} \circ \theta$ is not intrinsically equivalent to the identity map. Basically, this architecture only learns the Poisson map limited on a submanifold embedded in the whole phase space.

In both cases we perform predictions by $f_{PNN}^k = \theta^{-1} \circ \Phi^k \circ \theta$, which gives the kth step. We prefer the invertible neural networks because the reconstruction loss will disappear compared to the autoencoder, and we also expect to impose further prior information on the transformation. For example, in Section 3.4.3, we adopt a volume-preserving network as the invertible neural network, the so-called volume-preserving Poisson neural network (VP-PNN), which achieves better generalization compared to the non-volume-preserving Poisson neural network (NVP-PNN), since the original Poisson system has a volume-preserving phase flow. More crucially, autoencoder-based PNNs are unable to learn data lying on the whole space, rather than a $2d$-dimensional submanifold, when $2d < n$. In particular, the coordinate transformation twists the whole space and flatten a series of $2d$-dimensional submanifolds to planes, while the autoencoder only flattens a single $2d$-dimensional submanifold since it loses many degrees of freedom upon dimension reduction. However, autoencoder-based PNNs can perform better in some situations, such as in the numerical case in Section 3.4.5. Intuitively, the alternative architecture outperforms the primary architecture

when $2d \ll n$. An illustration of PNNs is presented in Fig. 3.1.

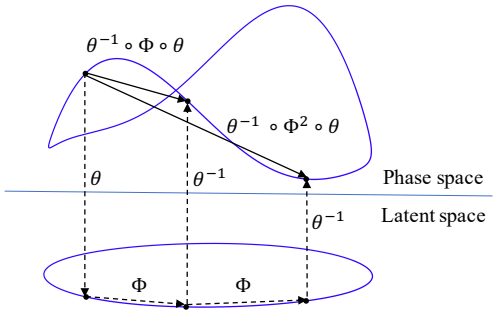


Figure 3.1: **Architecture of PNNs.** PNNs are composed of three parts: (1) a coordinate transformation, (2) an extended symplectic map, and (3) the inverse of the transformation, denoted by θ, Φ, and θ^{-1} respectively. θ and θ^{-1} are implemented by invertible neural networks for the primary architecture and by autoencoder for the alternative architecture, while Φ is implemented by the extended symplectic neural network, see Appendix 3.5 for details. The dimension of the latent space is equal to the original phase space, and the key difference is that the transformed system in latent space is "piecewise" Hamiltonian of latent dimension $2d$ while the whole space is of dimension $n \geq 2d$.

3.3.2 Learning Poisson systems

Consider the case that $\mathcal{T}$ consists of data points from a Poisson system. Next, we present the approximation properties of PNNs.

Theorem 10. Suppose that (i) the extended symplectic neural networks Φ are universal approximators within the space of continuously differentiable extended symplectic maps in C^1 topology, (ii) (primary architecture) the invertible neural networks θ are universal approximators within the space of invertible continuously differentiable maps which have non-degenerate Jacobian in the C^1 topology, and (iii) (alternative architecture) the transformation neural networks θ and θ^{-1} are universal approximators within the space of continuous maps in the C^0 topology. Then, the corresponding Poisson neural networks $\theta^{-1} \circ \Phi \circ \theta$ are able to approximate

arbitrary Poisson maps in C^1 topology for the primary architecture, and are able to approximate arbitrary Poisson maps (limited on submanifolds) within the space of continuous maps in C^0 topology for the alternative architecture. The approximations are considered on compact sets.

Proof. It can be deduced directly from Theorem 9 and Corollary 3. □

Notice that in the alternative architecture, the PNN $\theta^{-1} \circ \Phi \circ \theta$ itself is not intrinsically a Poisson map, however, by using $f_{PNN}^k = \theta^{-1} \circ \Phi^k \circ \theta$ for multistep prediction, it can also preserve geometric structure and enjoy stable long term performance in practice.

3.3.3 Learning trajectories of autonomous systems

Now consider the case that $\mathcal{T}$ consists of a series of data points on a single trajectory (maybe not from a Poisson system), i.e, $\mathcal{T} = \{(x_{i-1}, x_i)\}_1^N$, where $\phi_h(x_{i-1}) = x_i$ for time step h and ϕ_t, which is the phase flow of an unknown autonomous system $\dot{y} = f(y)$. Unlike the theory for learning Poisson systems, the use of PNNs to learn autonomous flows is quite novel. The theories are driven by the observation that symplectic neural networks can learn a trajectory from a non-Hamiltonian system, see Section 3.4.1 for details. The next theorem reveals the internal mechanism.

Theorem 11. Suppose that $U \subset \mathbb{R}^2$ is a simply connected open set, and the periodic solution $y(t) \in U$ is from an autonomous dynamical system

$$\dot{y} = f(y), \quad y \in U,$$

then there exists a Hamiltonian $H(y)$, such that $y(t)$ also satisfies the Hamiltonian system

$$\dot{y} = J^{-1}\nabla H(y), \quad y \in U.$$

Proof. The proof can be found in Appendix 3.5. □

Based on above theorem, one may be able to apply symplectic neural networks to arbitrary periodic solution to autonomous system in $\mathbb{R}^2$. Naturally, we tend to explore similar results in high-dimensional space. We intuitively expect to transform any high-dimensional periodic solution to autonomous system into one lying on a plane with the help of coordinate changes, and subsequently, the original trajectory can be learned via PNN. In fact, this conjecture is almost right, except for the case when the orbit of the considered motion is a non-trivial 1-knot in $\mathbb{R}^3$. For the theory on knots we refer to [128, 7, 159], and we briefly present the basic concepts in Appendix 3.5.

Theorem 12. Suppose that $U \subset \mathbb{R}^n$ is a contractible open set, periodic solution $y(t) \in U$ is from an autonomous dynamical system

$$\dot{y} = f(y), \quad y \in U,$$

and the orbit of $y(t)$ is unknotted. Then, there exists a Hamiltonian $H(y)$ and a $B(y)$ satisfying Lemma 5 with rank of 2, such that $y(t)$ also satisfies the Poisson system

$$\dot{y} = B(y)\nabla H(y), \quad y \in U.$$

Note that non-trivial 1-knots exist only in $\mathbb{R}^3$.

Proof. The proof can be found in Appendix 3.5. □

Up to now, we have shown that PNNs can be used to learn almost any periodic solution to autonomous systems, and the latent dimension is actually fixed as 2. The symplectic structure embedded in PNNs will endow the predictions with long term stability and more accuracy, for periodic solutions. Nevertheless, there is still a limitation of this method, as one can see, PNNs are allowed to learn only a single trajectory upon training. Basically, the limitation is inevitable as we have already got rid of most of the constraints on the vector field f, which is in fact a trade-off between data and systems. In spite of this fact, we still expect to further develop a strategy to relax the requirements of "single" and "periodic", by increasing the latent dimension. We conjecture it as:

Suppose that $U \subset \mathbb{R}^n$ is a contractible open set, $f : U \to \mathbb{R}^n$ is a vector field, and S is a smooth trivial $(2d-1)$-knot embedded in U. If $f|_S$ is a smooth tangent vector field on S, then there exists a smooth single-valued function $H(y)$ and a smooth matrix-valued function $B(y)$ satisfying Lemma 5 with rank of $2d$, such that $B(y)\nabla H(y)|_S = f|_S$.

The conjecture provides a more general insight into the above theoretical results on single trajectory. If it holds, one may learn several trajectories lying on a higher-dimensional trivial knot simultaneously upon training, with higher latent dimension. Unfortunately, the proofs of 1-knot case cannot be easily extended to the general case, since the fact that a solenoidal vector field on $\mathbb{R}^2$ is exactly a field of Hamiltonian system does not hold for higher-dimensional space. A more thorough explanation of this conjecture is needed in future works.

3.4 Simulation results

In this section, we present several simulation cases to verify our theoretical results, and indicate the potential application of PNNs in the field of computer vision. All the hyper-parameters for detailed architectures and training settings are shown in Appendix 3.5 and Table 3.3. For each detailed Poisson system involved, we obtain the ground truth and data using a high order symplectic integrator with its corresponding coordinate transformation, which is listed in Appendix 3.5. The codes are published in GitHub (https://github.com/jpzxshi/pnn).

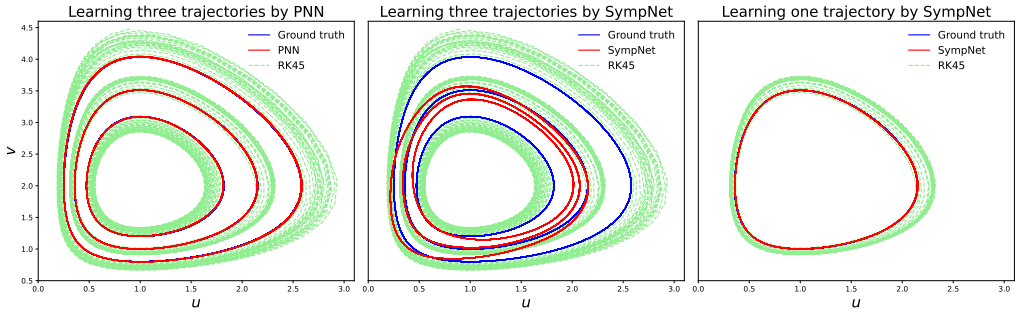


Figure 3.2: **Lotka–Volterra equation**. (**Left**) Learning three trajectories by PNN. The PNN successfully learns the system and achieves a stable long time prediction. We also compute the trajectories by the classical Runge-Kutta method of order four (RK45) to show that a general numerical method cannot guarantee stability. (**Middle**) Learning three trajectories by a SympNet. The SympNet fails to fit the three trajectories simultaneously, since the learned system is not Hamiltonian. (**Right**) Learning a single trajectory by a SympNet. The SympNet is able to learn this single trajectory due to Theorem 11.

3.4.1 Lotka–Volterra equation

The Lotka–Volterra equation can be written as

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} u(v-2) \\ v(1-u) \end{pmatrix} = \begin{pmatrix} 0 & uv \\ -uv & 0 \end{pmatrix} \nabla H(u,v), \tag{3.6}$$

where $H(u,v) = u - \ln u + v - 2\ln v$. As a Poisson system, we are able to discover the underlying symplectic structure using PNNs. The data consist of three trajectories, starting at $(1, 0.8), (1, 1), (1, 1.2)$, respectively. We generate 100 training points with time step $h = 0.1$ for each trajectory. Besides the PNN, we also use a SympNet to learn the three trajectories simultaneously, as well as learn the single trajectory starting at $(1, 1)$. We perform predictions for 1000 steps starting at the end points of the training trajectories, and the results of the three cases are presented in Fig. 3.2. As shown in the left figure, the PNN successfully learns the system and achieves a stable long time prediction. We also compute the trajectories by the classical Runge-Kutta method of order four (RK45) to show that solving the given system via a general numerical method cannot guarantee stability. Meanwhile, the SympNet [102] fails to fit the three trajectories simultaneously, since the data points are not from a Hamiltonian system, as shown in the middle figure. However, the right figure reveals that the SympNet is indeed able to learn a single trajectory, even though it is not from a Hamiltonian system, which is consistent with Theorem 11.

3.4.2 Extended pendulum system

We test the performance of a PNN on odd-dimensional Poisson systems. The motion of the pendulum system is governed by

$$\begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} -\sin q \\ p \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \nabla H(p,q),$$

where $H(p,q) = \frac{1}{2}p^2 - \cos q$. This is a canonical Hamiltonian system, and we subsequently extend this system to three-dimensional space:

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{c} \end{pmatrix} = \begin{pmatrix} -\sin q \\ p + c \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \nabla \tilde{H}(p,q,c),$$

where $\tilde{H}(p,q,c) = H(p,q) + pc$. To make the data more difficult to learn, a nonlinear transformation $(u,v,r) = \theta^{-1}(p,q,c) = (p, q, p^2 + q^2 + c)$ is applied to the extended phase space. The governing equation for the transformed system is as follows:

$$\begin{aligned} \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{pmatrix} &= \begin{pmatrix} 0 & -1 & -2v \\ 1 & 0 & 2u \\ 2v & -2u & 0 \end{pmatrix} \begin{pmatrix} u - 3u^2 - v^2 + r \\ \sin v - 2uv \\ u \end{pmatrix} \\ &=: B(u,v,r)\nabla K(u,v,r), \end{aligned} \tag{3.7}$$

where $K(u,v,r) = \frac{1}{2}u^2 - \cos v + ur - u^3 - uv^2$. One may readily verify that B satisfies Lemma 5, therefore (3.7) is a Poisson system.

Three trajectories are simulated with initial conditions $x_0 = (0, 1, 1^2)$, $(0, 1.5, 1.5^2 + 0.1)$, $(0, 2, 2^2 + 0.2)$, and time step $h = 0.1$. We use data points obtained in the first 100 steps as our training set. Then we perform predictions for 1000 steps starting at the end points of training set; the results are shown in Fig. 3.3. From the left figure, it can be seen that the predictions made by the PNN match the ground truth perfectly, remaining on the true trajectories after long times. Meanwhile, the PNN is able to recover the underlying structure of the system, as shown on the right hand side. The trajectories of the system in the latent space are recovered as trajectories on parallel planes, which matches the fact that the trajectories are generated from several different two-dimensional symplectic submanifolds.

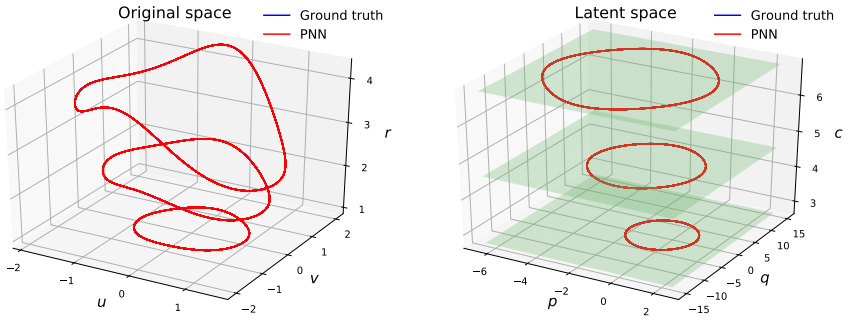


Figure 3.3: **Extended pendulum system.** (**Left**) Three ground truth trajectories compared to the predictions made by the PNN. The predictions match the ground truth perfectly without deviation. (**Right**) The ground truth and the predictions expressed in the latent space by trained θ. The trajectories are lying on three different parallel planes.

3.4.3 Charged particle in an electromagnetic potential

We consider the dynamics of the charged particle in an electromagnetic potential governed by the Lorentz force

$$m\ddot{x} = q(E + \dot{x} \times B),$$

where m is the mass, $x \in \mathbb{R}^3$ denotes the particle's position, q is the electric charge, $B = \nabla \times A$ denotes the magnetic field, and $E = -\nabla\varphi$ is the electric field with A, φ being the potentials. Let $\dot{x} = v$ be the velocity of the charged particle, then the governing equations of the particle's motion can be expressed as

$$\begin{aligned} \begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} &= \begin{pmatrix} -\frac{q}{m^2}\hat{B}(x) & -\frac{1}{m}I \\ \frac{1}{m}I & 0 \end{pmatrix} \nabla H(v, x), \\ H(v, x) &= \frac{1}{2} m v^\top v + q\varphi(x), \end{aligned} \tag{3.8}$$

where

$$\hat{B}(x) = \begin{pmatrix} 0 & -B_3(x) & B_2(x) \\ B_3(x) & 0 & -B_1(x) \\ -B_2(x) & B_1(x) & 0 \end{pmatrix}$$

for $B(x) = (B_1(x), B_2(x), B_3(x))$. Here we test the dynamics with $m = 1$, $q = 1$, and

$$A(x) = \frac{1}{3}\sqrt{x_1^2 + x_2^2} \cdot (-x_2, x_1, 0), \quad \varphi(x) = \frac{1}{100\sqrt{x_1^2 + x_2^2}}$$

for $x = (x_1, x_2, x_3)^\top$. Then

$$B(x) = (\nabla \times A)(x) = (0, 0, \sqrt{x_1^2 + x_2^2}),$$
$$E(x) = -(\nabla\varphi)(x) = \frac{(x_1, x_2, 0)}{100(x_1^2 + x_2^2)^{\frac{3}{2}}}.$$

The initial state is chosen to be $v = (1, 0.5, 0)$, $x = (0.5, 1, 0)$, in which case the system degenerates into four-dimensional dynamics, i.e., the motion of the particle is always on a plane. For simplicity, we also denote them by $v = (1, 0.5)$, $x = (0.5, 1)$, and study the dimension-reduced system. We then generate a trajectory of 1500 training points followed by 300 test points with time step $h = 0.1$. Subsequently, a volume-preserving PNN (VP-PNN) is trained to learn the training set, and we perform predictions for 2000 steps starting at the end point of the training set, as shown in Fig. 3.4. It can be seen that the VP-PNN perfectly predicts the trajectory without deviation. Furthermore, we also train a non-volume-preserving PNN (NVP-PNN) and a volume-preserving neural network (VPNN) to compare with the above model. After sufficient training, the three models make predictions starting at the initial state to reconstruct trajectories, as shown in Fig. 3.5. As one can see, the VP-PNN performs slightly better than the NVP-PNN, while the NVP-PNN is much better than the VPNN. The quantitative results shown in Table 3.1 also support this

observation. Although the VP-PNN has larger training MSE and one step test MSE than the NVP-PNN, its long time test MSE is instead less, which is not surprising because NVP-PNNs and VPNNs possess the prior information of Poisson structure and volume preservation respectively, while VP-PNNs has both of them. Note that the considered dimension-reduced system of (3.8) is source-free hence its phase flow is intrinsically volume-preserving on the four-dimensional space.

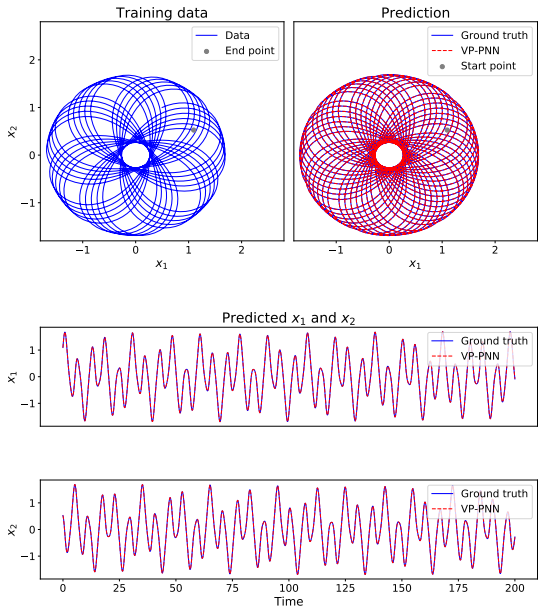


Figure 3.4: **Charged particle in the electromagnetic potential.** (**Top-left**) The position (x_1, x_2) of training flow. (**Top-right**) Prediction of the VP-PNN starting at the end point of training flow for 2000 steps. The VP-PNN perfectly predicts the trajectory without deviation. (**Bottom**) Prediction of the position of particle over time.

	VP-PNN	NVP-PNN	VPNN
Training MSE	4.6×10^{-9}	1.6×10^{-9}	2.7×10^{-8}
Test MSE (One step)	1.5×10^{-8}	9.8×10^{-9}	6.1×10^{-8}
Test MSE (Long time)	1.1×10^{-5}	5.1×10^{-4}	8.8×10^{3}

Table 3.1: **Losses of the three trained models.** The long time test MSE is evaluated along 1000 steps starting at the end point of training data. Although VP-PNN has larger training MSE and one step test MSE than NVP-PNN, its long time test MSE is instead less. The VPNN performs worse than above two models and fails for long time prediction.

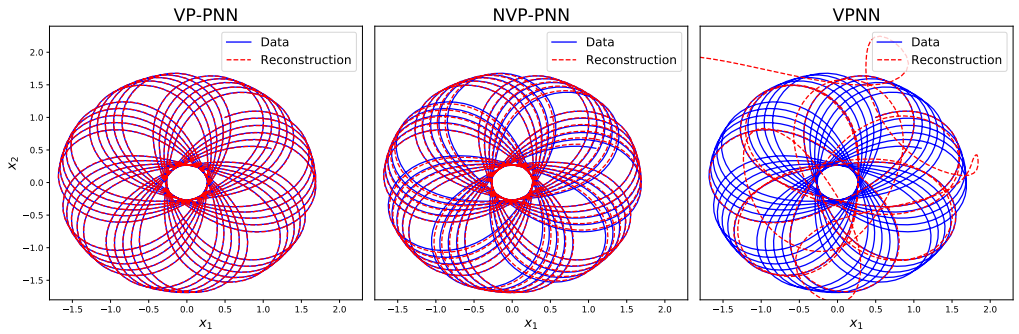


Figure 3.5: **Comparison of the reconstructed trajectories by three models.** (**Left**) The reconstructed trajectory by the VP-PNN, which perfectly matches the ground truth. (**Middle**) The reconstructed trajectory by the NVP-PNN, which performs well, but slightly worse than the VP-PNN. (**Right**) The reconstructed trajectory by the general VPNN, which fails to make long time prediction.

3.4.4 Nonlinear Schrödinger equation

We consider the nonlinear Schrödinger equation

$$\begin{cases} i\dfrac{\partial w}{\partial t} + \dfrac{\partial^2 w}{\partial x^2} + 2|w|^2 w = 0, \\ w(x,0) = w_0(x), \end{cases}$$

where $w(x,t) = u(x,t) + iv(x,t)$ is a complex field and the boundary condition $w_0(x)$ is periodic, i.e., $w_0(x+1) = w_0(x)$ for $x \in \mathbb{R}$. An interesting space discretization of the nonlinear Schrödinger equation is the Ablowitz–Ladik model

$$i\dot{w}_k + \frac{1}{\Delta x^2}(w_{k+1} - 2w_k + w_{k-1}) + |w_k^2|(w_{k+1} + w_{k-1}) = 0$$

with $w_k = w(k\Delta x, t)$, $\Delta x = 1/N$. Letting $w_k = u_k + iv_k$, we obtain

$$\begin{aligned} \dot{u_k} &= -\frac{1}{\Delta x^2}(v_{k+1} - 2v_k + v_{k-1}) - (u_k^2 + v_k^2)(v_{k+1} + v_{k-1}), \\ \dot{v_k} &= \frac{1}{\Delta x^2}(u_{k+1} - 2u_k + u_{k-1}) + (u_k^2 + v_k^2)(u_{k+1} + u_{k-1}). \end{aligned}$$

With $u = (u_1, \cdots, u_N)$, $v = (v_1, \cdots, v_N)$ this system can be written as

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & -D(u,v) \\ D(u,v) & 0 \end{pmatrix} \nabla H(u,v) \tag{3.9}$$

where $D = diag(d_1, \cdots, d_N)$ is the diagonal matrix with entries $d_k(u,v) = 1 + \Delta x^2(u_k^2 + v_k^2)$, and the Hamiltonian is

$$\begin{aligned} H(u,v) = &\frac{1}{\Delta x^2} \sum_{l=1}^{N} (u_l u_{l-1} + v_l v_{l-1}) \\ &- \frac{1}{\Delta x^4} \sum_{l=1}^{N} \ln(1 + \Delta x^2(u_l^2 + v_l^2)). \end{aligned}$$

We thus get a Poisson system. In the experiment, we choose the boundary condition $u(x,0) = 2 + 0.2 \cdot \cos(2\pi x)$, $v(x,0) = 0$ and set $N = 20$, hence (3.9) is a Poisson system of dimension 40. We then generate 500 training points followed by 100 test points with time step $h = 0.01$. That means the solution to this equation during the time interval $[0, 5]$ is treated as the training set, and then we learn the data using a PNN and predict the solution between $[5, 6]$. The result is shown in Fig. 3.6: both of the real part and imaginary part match the ground truth well.

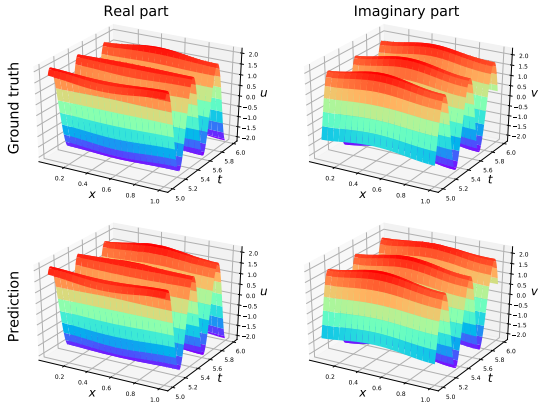


Figure 3.6: **Ablowitz–Ladik model of nonlinear Schrödinger equation.** Predictions of both real and imaginary parts match the ground truth well.

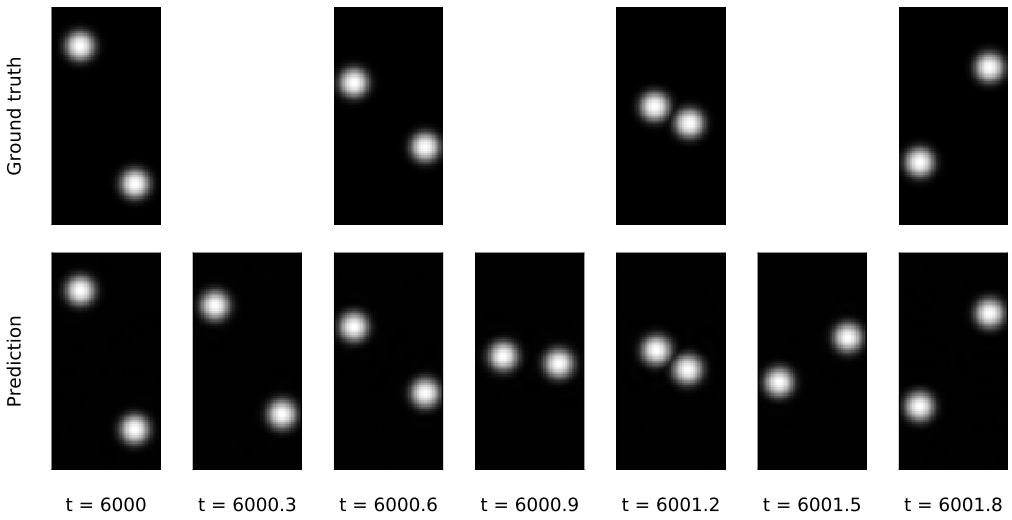


Figure 3.7: **Long time prediction and frame interpolation for two body images.** (**Top**) Ground truth, four consecutive points in the test dataset, after $t = 6000$ with step size $h = 0.6$. (**Bottom**) Predictions made by the PNN, with a finer step size $h = 0.3$. (**All**) PNNs can handle long-time integration and frame interpolation perfectly.

3.4.5 Pixel observations of two-body problem

We consider the pixel observations of the two-body problem, which are the images of two balls in black background, as shown in Fig. 3.7. Time series of the images form a movie of the motion of two balls governed by gravitation. Here we intend to learn the phase flow on a coarse time grid while making predictions on a finer time grid, to forecast and smoothen the movie simultaneously. To achieve this goal, a simple recurrent training scheme is applied to our method. Similar treatments can be found in [6].

Suppose the training dataset is $\{x_n := x(n\Delta t)\}_{n=0}^{N}$. We set our goal to be making predictions on $x(t)$ at $t = (N + \frac{n}{m})\Delta t$, $n \geq 1$. Denote the PNN to be trained as $f_{PNN} = \theta^{-1} \circ \Phi \circ \theta$. Then we train

$$f_{PNN}^{m} = \theta^{-1} \circ \Phi^{m} \circ \theta$$

to approximate the flow from x_i to x_{i+1}, and m is set to 2 in this case. Since we are learning a single trajectory on a submanifold of the high-dimensional space, autoencoders are used to approximate θ. λ in the corresponding loss function is chosen to be 1. After training, $f_{PNN}^k = \theta^{-1} \circ \Phi^k \circ \theta$ is used to generate predictions for $k \geq 1$.

A single trajectory $x(t)$ of the system is generated with time step $\Delta t = 0.6$, as shown in Fig. 3.7. The training dataset contains $N = 100$ images of size 100×50. Let $X_{grid} = (x_{N+1}, \cdots, x_{N+k})$ and $\tilde{X}_{grid} = (\tilde{x}_{N+1}, \cdots, \tilde{x}_{N+k})$ denote the ground truth and predictions made by the PNN at grid points. Let $X_{mid} = (x_{N+\frac{1}{2}}, \cdots, x_{N+k-\frac{1}{2}})$ and $\tilde{X}_{mid} = (\tilde{x}_{N+1}, \cdots, \tilde{x}_{N+k})$ denote the ground truth and predictions made by the PNN on middle points of the grids. The test loss on grids is calculated as the mean squared error between X_{grid} and $\tilde{X}_{grid}$ while the test loss on middle points is calculated as the mean squared error between X_{mid} and $\tilde{X}_{mid}$. We use a similar definition as in [188] to compute the valid prediction time T_ϵ. Suppose we are given the ground truth dataset x and prediction $\tilde{x}$, starting from $t = 0$. Let the root mean square error (RMSE) be

$$\mathcal{E}(\tilde{x}) = \sqrt{\langle (x - \tilde{x})^2 \rangle},$$

where $< \cdot >$ stands for spatial average. The valid prediction time (VPT) is defined to be

$$T_\epsilon = \arg\max_{t_f} \{ t_f | \mathcal{E}(\tilde{x}(t)) \leq \epsilon, \forall t \leq t_f \},$$

where ϵ is a hyperparameter to be chosen. Here we set $\epsilon = 0.02$.

Low error is obtained both on the grid points and the middle points, as shown in Table 3.2, which indicates that PNNs can handle prediction and interpolation simultaneously. The VPT is much longer than the time scale of the training window, further suggesting that PNNs are good at long-time predictions and intrinsically

structure-preserving. It can be seen in Fig. 3.7 that the prediction matches the ground truth perfectly even after $t = 6000$. We also compare the performance of AE-HNN and PNN in Fig. 3.8. The experimental details can be found in Appendix 3.5. We find that PNNs significantly outperform AE-HNNs in terms of prediction accuracy due to its structure-preserving nature.

Note that according to Theorem 12, the periodic solution to an autonomous dynamical system in $\mathbb{R}^d$ can always be learned by PNNs when $d > 3$. Since this trajectory of two-body system is periodic in $\mathbb{R}^{100\times 50}$ and the image at step $n+1$ is uniquely determined by the image at step n, we can assume without loss of generality that the pixel observations of a two-body system form a periodic solution to an autonomous dynamical system, regardless of whether the internal mechanism is Hamiltonian.

Train MSE	Test MSE (Grid)	Test MSE (Middle)	VPT
1.7×10^{-6}	2.1×10^{-6}	2.0×10^{-6}	6308

Table 3.2: **Losses and VPT of the PNN.** The test MSE is evaluated along 100 steps starting at the end point of data.

3.5 Summary

The main contribution of this chapter is to provide a novel high-level network architecture, PNN, to learn and further extrapolate the phase flow of an arbitrary Poisson system while preserving its intrinsic Poisson structure. To the best of our knowledge, this is the first work which can achieve this. Since a single periodic solution to an autonomous system can be proven to be a solution to a Poisson system if the orbit is unknotted, PNNs can be directly applied to a much broader class of systems without modification. From this perspective, theoretical results regarding the ap-

proximation ability of PNNs are presented. We present several simulations including the Lotka-Volterra equation, an extended pendulum system, charged particles in the electromagnetic potential, a nonlinear Schrödinger equation and a trajectory of the two-body image that support our theoretical findings and illustrate the advantages of PNNs on long time prediction and frame interpolation. Even though not explicitly mentioned in the chapter, PNNs can be easily extended to learn phase flows from irregularly sampled data using the feature of SympNets. PNNs can also learn the Hamiltonian systems on low dimensional submanifolds or constrained Hamiltonian systems, which can be expressed as a Poisson system. Note that PNN is very flexible as a high-level architecture, and one may use other preferred symplectic neural networks or invertible neural networks as alternative approximators if needed.

Despite the great expressivity, stability and interpretability of PNNs, an open issue is whether one can use it to infer and make predictions on multiple trajectories of an autonomous system without generalized Poisson structure under certain circumstances. We conjectured in the chapter that the solution to an arbitrary autonomous system lying on a smooth trivial $(2d-1)$-knot matches the solution to a Poisson system. If this holds, the use of PNNs on multiple trajectories of a general autonomous system would be theoretically justified. We leave the proof or counterexamples of this conjecture as future work.

Appendix

Introduction to knot

Consider the embeddings of S^n in S^m, $m \geq n+1$. Two embeddings k_1, k_2 are equivalent if there is a homeomorphism h of S^m such that $h(k_1) = k_2$. An embedding $k : S^n \subset S^m$ is *unknotted* if it is equivalent to the trivial knot $k_0 : S^n \subset S^m$ defined by the standard embedding

$$k_0 : S^n \to S^m;\ (x_0, \cdots, x_n) \to (x_0, \cdots, x_n, 0, \cdots, 0).$$

In fact, the embedding k is always unknotted when $m \neq n+2$. Therefore an n-knot is defined as an embedding of S^n in S^{n+2}. For convenience, let us make a little change in the terminology: an n-knot means an embedding of S^n in S^m for $m \geq n+1$, since we are more concerned about the trivial knot in this work.

Proofs for theorems

Proof of Theorem 11

In two-dimensional space, consider the system

$$\begin{cases} \Delta \boldsymbol{v} = \text{grad } p, \\ \text{div } \boldsymbol{v} = 0, \end{cases}$$

with boundary condition

$$\boldsymbol{v}|_\Gamma = \boldsymbol{a},$$

where Γ is the orbit of $y(t)$ and $\boldsymbol{v} : U \to \mathbb{R}^2$ is a vector field. [116, p. 60, Theorem 1] shows that there exists a solution $\boldsymbol{v}$ to the system above given a suitable single-valued function p and any continuous $\boldsymbol{a}$ satisfying

$$\oint_\Gamma \boldsymbol{a} \cdot \boldsymbol{n} ds = 0,$$

where $\boldsymbol{n}$ is the exterior normal with respect to the domain inside Γ. Set $\boldsymbol{a} = f$, we have

$$\oint_\Gamma f \cdot \boldsymbol{n} ds = \oint_\Gamma 0 ds = 0,$$

hence there exists a $\boldsymbol{v}$ such that

$$\text{div } \boldsymbol{v} = 0, \quad \boldsymbol{v}|_\Gamma = f.$$

Note that $\boldsymbol{v}$ is a solenoidal vector field, which leads to

$$\boldsymbol{v} = \text{curl } (-H) = (-\frac{\partial H}{\partial y_2}, \frac{\partial H}{\partial y_1})^\top = J^{-1} \nabla H.$$

The H defined above is exactly the Hamiltonian we are looking for.

Proof of Theorem 12

Since $y(t)$ is unknotted, there exist a periodic $\gamma(t) = (\gamma_1(t), \gamma_2(t))^\top \in \mathbb{R}^2$ and a homeomorphism $\theta : U \to U$ such that

$$\theta(y(t)) = \Gamma(t) := (\gamma_1(t), \gamma_2(t), 0, \cdots, 0)^\top \in \mathbb{R}^n,$$

which is exactly the coordinate transformation deforming $y(t)$ into $\Gamma(t)$. Theorem 11 shows that $\gamma(t)$ is also a solution to a Hamiltonian system, hence the extended solution $\Gamma(t)$ satisfies

$$\begin{cases} \dot{\Gamma} = \begin{pmatrix} J^{-1} & 0 \\ 0 & 0 \end{pmatrix} \nabla K(\Gamma), \quad J \in \mathbb{R}^{2\times 2}, \\ K(y_1, y_2, \cdots, y_n) = H(y_1, y_2), \end{cases} \tag{3.10}$$

for a single-valued function H. In fact, a Poisson system expressed in an arbitrary new coordinate immediately becomes a new Poisson system with a new $B(y)$ whose rank is equivalent to the original one [84, p. 265]. Therefore, $y(t)$ is also a solution to a Poisson system obtained by expressing system (3.10) in new coordinate via transformation θ. Furthermore, they share the same latent dimension of 2.

Implementation of architecture

Problem		Section 3.4.1		Section 3.4.2	Section 3.4.3		Section 3.4.4	Section 3.4.5
		PNN	SympNet		PNN	VPNN		
θ	Type	NVP	-	NVP	VP/NVP	VP	NVP	AE
	Partition	1	-	2	2	2	20	-
	Layers	3	-	3	10	20	3	2
	Sublayers	2	-	2	3	3	2	-
	Width	30	-	30	50	50	100	50
Φ	Type	G	G	E	G	-	G	LA
	Layers	3	6	3	10	-	10	3
	Sublayers	-	-	-	-	-	-	2
	Width	30	30	30	50	-	100	-

Table 3.3: **Model architecture.** The detailed implementations of symplectic neural networks, invertible neural networks as well as the autoencoder, and their corresponding terminology used in this table are shown in Appendix 3.5. The Sigmoid activation functions are used for all the models. The optimizer is chosen to be Adam [108] with learning rate 0.001. The numbers of training iterations are 2×10^5, 1×10^5, 2×10^6, 1×10^6, and 5×10^5 respectively for the five problems considered.

Extended symplectic neural networks

We adopt SympNets [102] as the universal approximators for symplectic maps. The architecture of SympNets is based on three modules, i.e.,

- **Linear modules.**

$$\begin{aligned}&\mathcal{L}_n\begin{pmatrix}p\\q\end{pmatrix}\\&=\begin{pmatrix}I & 0/S_n\\ S_n/0 & I\end{pmatrix}\cdots\begin{pmatrix}I & 0\\ S_2 & I\end{pmatrix}\begin{pmatrix}I & S_1\\ 0 & I\end{pmatrix}\begin{pmatrix}p\\q\end{pmatrix}+b,\\&p,q\in\mathbb{R}^d,\end{aligned}$$

  where $S_i \in \mathbb{R}^{d\times d}$ are symmetric, $b \in \mathbb{R}^{2d}$ is the bias, while the unit upper triangular symplectic matrices and the unit lower triangular symplectic matrices appear alternately. In this module, S_i (represented by $A_i + A_i^\top$ in practice) and b are parameters to learn. In fact, $\mathcal{L}_n$ can represent any linear symplectic map [100].

- **Activation modules.**

$$\begin{aligned}\mathcal{N}_{up}\begin{pmatrix}p\\q\end{pmatrix}&=\begin{pmatrix}p+\tilde{\sigma}_a(q)\\q\end{pmatrix},\\ \mathcal{N}_{low}\begin{pmatrix}p\\q\end{pmatrix}&=\begin{pmatrix}p\\\tilde{\sigma}_a(p)+q\end{pmatrix},\quad p,q\in\mathbb{R}^d,\end{aligned}$$

  where $\tilde{\sigma}_a(x) := a \odot \sigma(x)$ for $x \in \mathbb{R}^d$. Here $\odot$ is the element-wise product, σ is the activation function, and $a \in \mathbb{R}^d$ is the parameter to learn.

- **Gradient modules.**

$$\mathcal{G}_{up}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_{K,a,b}(q) \\ q \end{pmatrix},$$
$$\mathcal{G}_{low}\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_{K,a,b}(p) + q \end{pmatrix}, \quad p, q \in \mathbb{R}^d,$$

  where $\hat{\sigma}_{K,a,b}(x) := K^\top(a \odot \sigma(Kx + b))$ for $x \in \mathbb{R}^d$. Here $a, b \in \mathbb{R}^l$, $K \in \mathbb{R}^{l \times d}$ are the parameters to learn, and l is a positive integer regarded as the width of the module.

The SympNets are the composition of above three modules. In particular, we use two classes of SympNets: the LA-SympNets composed of linear and activation modules, and the G-SympNets composed of gradient modules. Notice that both LA and G SympNets are universal approximators for symplectic maps as shown in [102]. For convenience, we clarify the terminology for describing a detailed LA(G)-SympNet: an LA-SympNet of m layers with n sublayers means it is the composition of m linear modules and $m-1$ activation modules, where linear and activation modules appear alternatively like the architecture of fully-connected neural network, and each linear module is composed of n alternated triangular symplectic matrices; a G-SympNet of m layers with width l means it is composed of m alternated gradient modules, and the width l is defined as above.

In this works we further develop the extended symplectic neural networks by extending the gradient modules.

- **Extended modules.**

$$\mathcal{E}_{up}\begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p + \widehat{\sigma}_{K_1,K_2,a,b}(q,c) \\ q \\ c \end{pmatrix},$$

$$\mathcal{E}_{low}\begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p \\ \widehat{\sigma}_{K_1,K_2,a,b}(p,c) + q \\ c \end{pmatrix},$$

$$p, q \in \mathbb{R}^d,\ c \in \mathbb{R}^{n-2d},$$

where $\widehat{\sigma}_{K_1,K_2,a,b}(x,c) := K_1^\top(a \odot \sigma(K_1 x + K_2 c + b))$ for $x \in \mathbb{R}^d$, $c \in \mathbb{R}^{n-2d}$. Here $a, b \in \mathbb{R}^l$, $K_1 \in \mathbb{R}^{l\times d}$, $K_2 \in \mathbb{R}^{l\times(n-2d)}$ are the parameters to learn, and l is a positive integer regarded as the width of the module.

The extended symplectic neural networks (E-SympNets) are the composition of extended modules. As noticed, $2d$ is the latent dimension of the E-SympNets. The terminology for describing the architecture of E-SympNets is the same as that for G-SympNets, hence we will not repeat here. It is worth mentioning that the approximation property of E-SympNets is still unknown, which is left as future work.

Invertible neural networks

In numerical experiments, we adopt the NICE [51] as volume-preserving invertible neural networks, and the real NVP [52] as general non-volume-preserving invertible neural networks. The approximation capabilities of NICE are analysed in [200].

- **Volume-preserving modules.**

$$\mathcal{V}_{up}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + m_1(x_2) \\ x_2 \end{pmatrix},$$
$$\mathcal{V}_{low}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ m_2(x_1) + x_2 \end{pmatrix},$$
$$x_1 \in \mathbb{R}^d, x_2 \in \mathbb{R}^{n-d},$$

  where $m_1 : \mathbb{R}^{n-d} \to \mathbb{R}^d$ and $m_2 : \mathbb{R}^d \to \mathbb{R}^{n-d}$ are modeled as fully-connected neural networks.

- **Non-volume-preserving modules.**

$$\mathcal{C}_{up}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \odot \exp s_1(x_2) + t_1(x_2) \\ x_2 \end{pmatrix},$$
$$\mathcal{C}_{low}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \odot \exp s_2(x_1) + t_2(x_1) \end{pmatrix},$$
$$x_1 \in \mathbb{R}^d, x_2 \in \mathbb{R}^{n-d},$$

  where $s_1, t_1 : \mathbb{R}^{n-d} \to \mathbb{R}^d$ and $s_2, t_2 : \mathbb{R}^d \to \mathbb{R}^{n-d}$ are modeled as fully-connected neural networks.

Basically, the volume-preserving (non-volume-preserving) invertible neural networks are the alternated composition of upper and lower volume-preserving (non-volume-preserving) modules. We say a volume-preserving (non-volume-preserving) INN is of m layers and n sublayers with width l if it is composed of m alternated volume-preserving (non-volume-preserving) modules and each of m_1, m_2 (s_1, t_1, s_2, t_2) is a FNN of n layers with width l. Meanwhile, the partition dimension d is also fixed

as an architecture parameter and, in practice, we intuitively set $d \approx n/2$ for more expressivity. Note that this d has nothing to do with the latent dimension of extended symplectic neural networks.

Autoencoder

We apply the traditional autoencoder [112] to the alternative architecture for dimension-reduced cases. Suppose that the latent dimension is $\operatorname{rank}(B(y)) = 2d < n$, then the aotuencoder is composed of two fully-connected neural networks, i.e., an encoder $f_e : \mathbb{R}^n \to \mathbb{R}^{2d}$ and a decoder $f_d : \mathbb{R}^{2d} \to \mathbb{R}^n$, whose hidden nodes are all not less than $2d$. Note that the FNNs can also be replaced by other useful architectures like CNNs, if needed. In Section 3.4, the encoder and decoder are chosen of same depth and width.

Transformations for Poisson systems

Here we briefly present the transformations to Hamiltonian systems for the involved Poisson systems in this work.

Lotka–Volterra equation. With coordinate transformation $(p, q) = (\ln u, \ln v)$, system (3.6) can be written as

$$\begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} = J^{-1} \nabla H(p, q), \quad H(p, q) = p - \exp(p) + 2q - \exp(q).$$

Charged particle in the electromagnetic potential. By considering the position x and the conjugate momentum $p = m\dot{x} + qA(x)$, the Poisson system (3.8) can

be written in the canonical form

$$\begin{pmatrix} \dot{p} \\ \dot{x} \end{pmatrix} = J^{-1}\nabla H(p,x),$$
$$H(p,x) = \frac{1}{2m}(p - qA(x))^\top (p - qA(x)) + q\varphi(x).$$

Nonlinear Schrödinger equation. A transformation has been proposed to make (3.9) a canonical Hamiltonian system:

$$\begin{cases} p_k = u_k \sigma(\Delta x^2(u_k^2 + v_k^2)) \\ q_k = v_k \sigma(\Delta x^2(u_k^2 + v_k^2)) \end{cases}, \quad \text{with} \quad \sigma(x) = \sqrt{\frac{\ln(1+x)}{x}},$$

which treats the variables symmetrically. Its inverse is

$$\begin{cases} u_k = p_k \tau(\Delta x^2(p_k^2 + q_k^2)) \\ v_k = q_k \tau(\Delta x^2(p_k^2 + q_k^2)) \end{cases}, \quad \text{with} \quad \tau(x) = \frac{\exp x - 1}{x}.$$

Now the Hamiltonian in the new variables is

$$\begin{aligned} H(p,q) =& \frac{1}{\Delta x^2} \sum_{l=1}^{N} \tau(\Delta x^2(p_l^2 + q_l^2))\tau(\Delta x^2(p_{l-1}^2 + q_{l-1}^2)) \\ & \cdot (p_l p_{l-1} + q_l q_{l-1}) - \frac{1}{\Delta x^2} \sum_{l=1}^{N} (p_l^2 + q_l^2). \end{aligned}$$

Comparison with baseline

We compare the performance between PNN and AE-HNN [81] on the two body image problem in the same setting as 3.4.5. The detailed architecture for PNN is already given in Table 3.3. For AE-HNN, the layers and width of the autoencoder

are chosen to be 4 and 200; while the layers and width of HNN are chosen to be 3 and 200. The latent dimension is chosen to be 4 and the activation function is chosen to be hyperbolic tangent. We find that PNN significantly outperforms AE-HNN in a few time steps, as shown in Fig. 3.8 below. The predictions of AE-HNN becomes blurry and do not capture the correct period of the system.

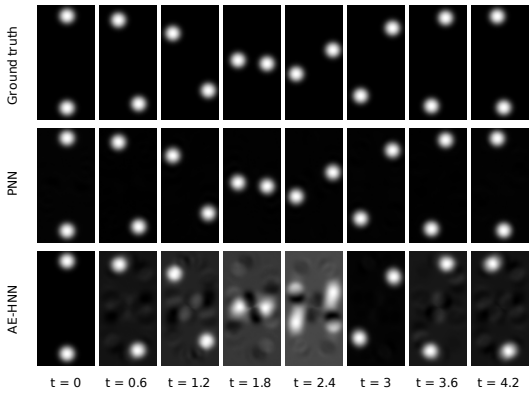


Figure 3.8: **Comparison between PNN and AE-HNN for two body images.** (**Top**) Ground truth, eight consecutive points in the dataset, after $t = 0$ with step size $h = 0.6$. (**Bottom**) Predictions made by the AE-HNNs, which become blurry after a few time steps and do not capture the correct period of the system. (**Middle**) Predictions made by the PNNs, which match the ground truth well.

Chapter Four

GFINNs: GENERIC Formalism Informed Neural Networks for Deterministic and Stochastic Dynamical Systems

4.1 Introduction

Among many principles of physics, we consider the General Equation for Non-Equilibrium Reversible-Irreversible Coupling (GENERIC) formalism [82, 150, 149]. GENERIC provides a general mathematical framework describing states beyond equilibrium of a dynamical system [149], which involves two separate generators for the reversible and irreversible dynamics. These generators are required to satisfy some symmetry and degeneracy conditions, which constitute a key feature of the GENERIC structure. These conditions are often interpreted as the first and second principles of thermodynamics, which further can be expressed in the language of linear algebra.

Our goal is to embed the GENERIC structure directly into neural networks, yet to maintain sufficient expressivity. By leveraging the level of prior physics information under the GENERIC framework, we propose a systematical approach in designing neural network modules.

In the case where either one or all generators are known *a priori*, we design neural network models for either energy or entropy or both by exploiting certain properties of the generators. Due to the multiplicative structure of the gradient of neural networks, care needs to be taken into the input layer in order to meet the degeneracy conditions. We thus introduce a transformation in the first layer, which roughly speaking, projects input into a proper low-dimensional space on which the degeneracy conditions hold. On the other hand, if no prior information is available, we model generators by neural networks whose architectures are motivated by the spectral decomposition of matrix [185]. In any cases, the proposed neural network models obey the required constraints exactly. We refer to our neural network models

as the GENERIC formalism informed neural networks (GFINNs).

Furthermore, we prove the universal approximation theorem for GFINNs under some assumptions. Altogether, GFINNs not only obey the required GENERIC conditions, but also are sufficiently expressive in learning the underlying physical quantities and generators. We demonstrate the performance of GFINNs on several tasks including double pendulums, gas containers and stochastic differential equations. We found that GFINNs outperform the existing network architectures [50, 87, 122] in all the tests we considered.

There are two pioneering works that have attempted to embed the GENERIC formalism into neural networks. [87] proposed structure preserving neural networks (SPNNs), which aim to learn the physical quantities assuming both generators are known. The degeneracy requirements are softly constrained in the loss function, which may cause violation of the degeneracy conditions even after training. [122] proposed the GENERIC neural ordinary differential equations (GNODEs), which satisfy the required conditions by suitable parameterization of bracket structure, however, no universal approximation theorem has been proven.

The rest of the chapter is organized as follows. Upon introducing the problem setup and some preliminaries in Section 4.2, the GENERIC formalism informed neural networks (GFINNs) are presented in Section 4.3 along with the universal approximation theorem. Numerical examples are provided in Section 4.4, demonstrating the effectiveness of GFINNs against other methods.

4.2 Problem Setup

We consider the problem of learning an autonomous dynamical system that can be expressed in the following form:

$$\dot{\boldsymbol{z}}(t) = F(\boldsymbol{z}(t)), \qquad \boldsymbol{z} \in \Omega \subset \mathbb{R}^d, \quad t \in (0, T], \quad \boldsymbol{z}(0) = \boldsymbol{z}_0, \tag{4.1}$$

where t refers to the time coordinates and F is the unknown vector-valued field function.

With the goal of approximating F through data, we employ a neural network $F_{\mathrm{NN}}(\cdot;\theta)$ whose parameters are determined so that the trajectories generated from the resulting dynamics $\dot{\boldsymbol{z}} = F_{\mathrm{NN}}(\boldsymbol{z};\theta)$ are close to the observed trajectory data. More precisely, let N_{traj} be the number of observed trajectories. Let T be the number of time-steps and t_j be the j-th time stamp, which we set them equal for all trajectories for the sake of notational simplicity. Let $\boldsymbol{z}(t;\boldsymbol{z}_0)$ be the state variable at time t whose value at t_0 is $\boldsymbol{z}_0$. Then, the k-th trajectory is written as $\{t_j, \boldsymbol{z}(t_j;\boldsymbol{z}_0^{(k)})\}_{j=0}^{T}$. We then seek to find the optimal network parameters that minimize the loss function defined by

$$\mathcal{L}(\theta) = \frac{1}{N_{\mathrm{traj}}} \sum_{k=1}^{N_{\mathrm{traj}}} \frac{1}{T} \sum_{j=1}^{T} \left\| \boldsymbol{z}_{\mathrm{NN}}^{\theta}(t_j;\boldsymbol{z}_0^{(k)}) - \boldsymbol{z}(t_j;\boldsymbol{z}_0^{(k)}) \right\|^2. \tag{4.2}$$

Here $\boldsymbol{z}_{\mathrm{NN}}^{\theta}(t_j;\boldsymbol{z}_0^{(k)})$ is computed by applying a numerical integrator [118] (e.g., Runge–Kutta methods) to the equation $\dot{\boldsymbol{z}} = F_{\mathrm{NN}}(\boldsymbol{z};\theta)$ starting at $\boldsymbol{z}(t_{j-1};\boldsymbol{z}_0^{(k)})$; $\|\cdot\|$ is the standard Euclidean norm. Typically, gradient-based optimization methods are used to solve this minimization problem.

If no physics is involved, the above gives a general description of the pure data-driven approach. If some principles of physics are known, the physics-informed data-

driven approach aims to embed the available physics into neural networks. There are two typical ways of doing the embedding. One is to add regularization terms to the loss (4.2) that penalize F_{NN} that do not follow the physics. The other is to devise a network architecture so that F_{NN} obeys the available physics for any θ and the optimal parameters are then found by minimizing the same loss (4.2).

4.2.1 The GENERIC formalism

The General Equation for Non-Equilibrium Reversible-Irreversible Coupling (GENERIC) formalism provides a general mathematical framework describing *beyond-equilibrium* thermodynamic systems [149] including both conservative and dissipative systems. As a consequence, any system described by Hamilton's equation or Poisson's equation can be written in the GENERIC formalism, as follows:

$$\begin{aligned} & \dot{\boldsymbol{z}} = L(\boldsymbol{z})\frac{\partial E}{\partial \boldsymbol{z}}(\boldsymbol{z}) + M(\boldsymbol{z})\frac{\partial S}{\partial \boldsymbol{z}}(\boldsymbol{z}), \\ \text{subject to} \quad & L(\boldsymbol{z})\frac{\partial S}{\partial \boldsymbol{z}}(\boldsymbol{z}) = M(\boldsymbol{z})\frac{\partial E}{\partial \boldsymbol{z}}(\boldsymbol{z}) = \boldsymbol{0}, \\ & L(\boldsymbol{z}) \text{ is skew-symmetric, i.e., } L(\boldsymbol{z}) = -L(\boldsymbol{z})^{\top} \\ & M(\boldsymbol{z}) \text{ is symmetric positive semi-definite.} \end{aligned} \tag{4.3}$$

The term $L\frac{\partial E}{\partial \boldsymbol{z}}$ accounts for all the reversible (non-dissipative) phenomena of the system. In the classical mechanics, this term is equivalent to Hamilton and Poisson's equations of motion. The operator L is called the Poisson matrix and is required to be skew-symmetric. The term $M\frac{\partial S}{\partial \boldsymbol{z}}$ accounts for the irreversible (dissipative) material properties of the system. This term was motivated by the Ginzburg-Landau equation, which can be used to describe critical dynamics of spatially extended sys-

tems. The operator M is called the friction matrix and is required to be symmetric positive semi-definite. E and S are the system's total energy and entropy, respectively. Under this framework, it can be checked that the energy of the system is conserved and the entropy of the system monotonically increases with respect to time, i.e., $\frac{dE(\boldsymbol{z}(t))}{dt} = 0$ and $\frac{dS(\boldsymbol{z}(t))}{dt} \geq 0$, corresponding to the first and second laws of thermodynamics, respectively.

4.2.2 Deep Neural Networks

We employ deep neural networks as the basic components in our surrogate modellings for L, M, E, S. For simplicity of discussion, we shall focus on feed-forward neural networks throughout this work, while any types of neural networks (e.g., ResNet) can easily be used in place of the feed-forward networks without difficulties.

A L-layer feed-forward neural network $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is defined by $f(x) = z^{L+1}(x)$ where z^{L+1} is constructed recursively according to

$$z^{\ell}(x) = W^{\ell}\phi(z^{\ell-1}(x)) + b^{\ell}, \qquad 1 < \ell \leq L+1,$$

starting with $z^1(x) = W^1 x + b^1$. Here, $W^{\ell} \in \mathbb{R}^{n_{\ell} \times n_{\ell-1}}$ is the weight matrix and $b^{\ell} \in \mathbb{R}^{n_{\ell}}$ is the bias vector in the l-th layer, where we set $n_0 = d_{\text{in}} = d$ and $n_L = d_{\text{out}}$. ϕ is a nonlinear activation function that is applied element-wise. The activation function is assumed to have certain properties so that the universal approximation theorem holds [37, 177, 143]. The collection of all weights and biases of the network is denoted by $\theta = \{(W^1, b^1), \cdots, (W^L, b^L)\}$. We note that a neural network can be a matrix-valued function by converting the output to a matrix of proper size.

Let $f(\cdot;\theta) : \mathbb{R}^d \to \mathbb{R}$ be a L-layer neural network and $g : \mathbb{R}^d \to \mathbb{R}^d$ be a differentiable function. The gradient of $(f \circ g)(x) := f(g(x))$ with respect to x is given by

$$\nabla_x f(g(x);\theta) = (\boldsymbol{J}g(x))^\top \left(W^L D_{L-1} \cdots W^2 D_1 W^1\right)^\top = (\boldsymbol{J}g(x))^\top \nabla_x f \circ g(x), \quad (4.4)$$

where D_j is a diagonal matrix whose (i,i)-entry is $\phi'(z_i^j(g(x)))$ for $1 \leq j < L$ and $1 \leq i \leq n_j$, and $\mathbf{J}g(x)$ is the Jacobian of g at x. A key observation is that $\nabla_x f(g(x);\theta)$ belongs to the row space of $\mathbf{J}g(x)$ due to the multiplicative structure. We refer to the equation (4.4) as the multiplicative structure of the gradient of neural networks.

4.3 GENERIC Formalism Informed Neural Networks

Our goal is to design neural network architectures that satisfy the symmetry and degeneracy conditions of (4.3), yet are sufficiently expressive to learn the underlying dynamics from data. Also, we want the proposed neural networks to be easily adopted when some prior physics is available (in terms of the GENERIC).

Since the GENERIC formalism comprises two orthogonal modules, each of which contains two components, we model each component using neural networks that satisfy the required conditions. The component-wise design not only allows the flexibility in incorporating prior physics (if any) but also results in a general framework of neural network modellings for the GENERIC formalism. Here, prior physics implies a scenario where one or more is known among L, M, E, S. Although many

possibilities can be discussed, for the sake of simplicity, we focus on the following scenarios:

- Case 1: L and M are known. The goal is to approximate E and S.
- Case 2a: E and S are known. The goal is to approximate L and M.
- Case 2b: L, M, E, S are unknown. The goal is to approximate L, M, E, S.

All other scenarios can be handled without difficulties. The schematic of the proposed framework is shown in Figure 4.1.

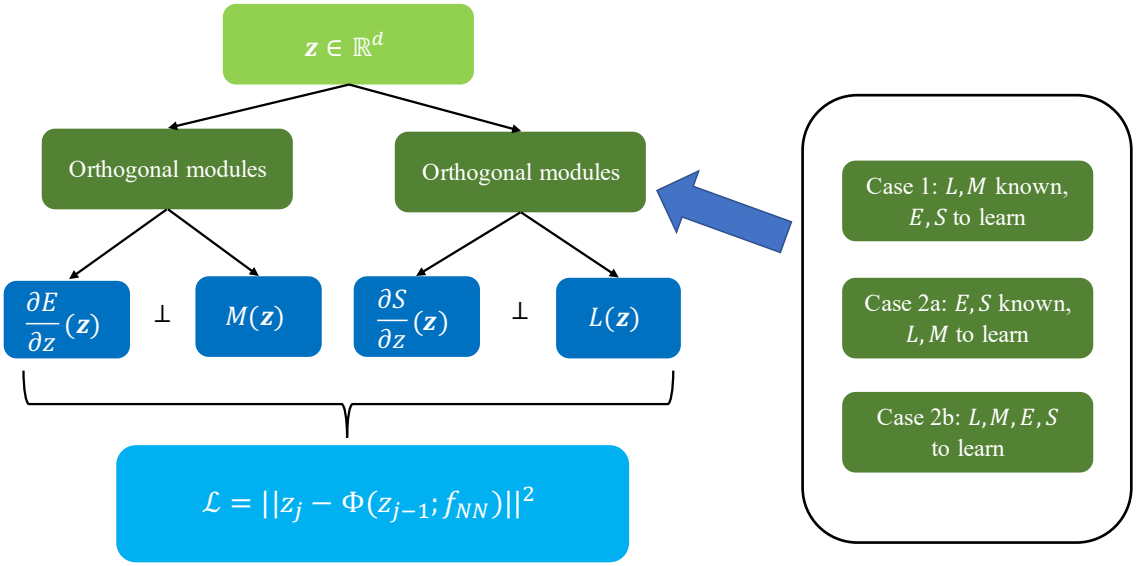


Figure 4.1: **Architecture of GFINNs.** GFINNs can be seen as a neural network framework, which automatically satisfies the laws of thermodynamics. The design of GFINNs follows the GENERIC formalism since we parameterize the (matrix-valued) functions L, M, E, S as orthogonal neural network modules, which satisfy the consistency condition and structural condition in Eq. 4.3. We propose different architectures, which can incorporate different physical information including the knowledge of L, M or the knowledge of E, S.

Requiring only the conditions of (4.3) can be done quite easily, while care is

needed to ensure expressivity. We illustrate this through the following example. Let

$$L(\boldsymbol{z}) = \begin{bmatrix} 0 & \ell(\boldsymbol{z}) & 0 & 0 \\ -\ell(\boldsymbol{z}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \ell(\boldsymbol{z}) \neq 0.$$

Suppose the goal is to construct a proper neural network S_{NN} satisfying $L(\boldsymbol{z})\nabla S_{\text{NN}}(\boldsymbol{z}) = 0$. Since $\text{null}(L(\boldsymbol{z})) = \text{span}\{e_3, e_4\}$, where $e_3 = (0, 0, 1, 0)^\top$ and $e_4 = (0, 0, 0, 1)^\top$, we have infinitely many choices for S_{NN} that satisfy the condition. For example, if S_{NN} is any smooth function of $\boldsymbol{z}_3$ and does not depend on the other variables, the required condition is trivially satisfied. However, this class of functions does not capture the potential dependence of $\boldsymbol{z}_4$, so that a sufficient expressiveness is not guaranteed. Similarly, if S were given and the goal was to model a skew-symmetric matrix L_{NN} that satisfies $L_{\text{NN}}(\boldsymbol{z})\nabla S(\boldsymbol{z}) = 0$, there are also infinitely many choices for L_{NN} that satisfy the required condition, yet not all of them are sufficiently expressive.

For ease of discussion, in what follows, we denote a generic matrix function by A representing either L or M, and a generic scalar function by G representing either S or E. Also, the subscript (G_{NN} or A_{NN}) indicates the neural network model for the target quantity (G or A). Next, we discuss how to construct neural networks for modelling each component and present the corresponding universal approximation theorem under some assumptions.

4.3.1 Case 1: L and M are known and E and S are unknown

We consider the case where L and M are known, yet E and S are unknown. Since the two generators are known, one might attempt to model ∇E and ∇S directly using neural networks following the pure data-driven approach [31]. However, since all vector functions are not gradient of a scalar function, we construct neural networks for E and S and compute their gradients by automatic differentiation that can be implemented by well-established programming packages e.g., Pytorch [153] and Tensorflow [1]. Furthermore, by construction, this approach allows one to not only predict the solution trajectories but also discover physical quantities (energy and entropy) from data.

Since we have multiple goals to achieve, several challenges arise in developing neural network architectures with the desired properties. First of all, since we model G (i.e., E or S) instead of ∇G, we need to properly control the gradient of neural networks so that the degeneracy condition of (4.3) holds. Motivated by the multiplicative structure (4.4), we introduce a tailored projection-like transformation $\mathcal{P}_A$ in the very first layer of neural networks, which ensures the degeneracy condition under some assumptions. It is the transformation $\mathcal{P}_A$ that constitutes a core element in the network architecture for G (either E and S) assuming A is known.

As a preparation for introducing the transformation $\mathcal{P}_A$ and also for the universal approximation theorem, we make a couple of basic assumptions on A, which will be justified in all the examples later.

Assumption 1. Let A be a $d \times d$ matrix-valued function defined on $\Omega \subset \mathbb{R}^d$. Let $q_A^j(\boldsymbol{z})$, $j = 1, \dots, n_A$, be an orthonormal basis of $\ker A(\boldsymbol{z}) := \{y \in \mathbb{R}^d : A(\boldsymbol{z})y = 0\}$ whose rank is n_A. We assume that

1. $\ker A(\boldsymbol{z})$ has constant rank $n_A \in \{1, \ldots, d-1\}$ in Ω.

2. $\ker_{\text{inv}} A(\boldsymbol{z}) := \text{span}\{q_A^j : 1 \leq j \leq \tilde{n}_A\}$ is the largest subspace of $\ker A(\boldsymbol{z})$ whose rank is $\tilde{n}_A$ which is also constant on Ω such that $q_A^j(\boldsymbol{z})$, $j = 1, \ldots, \tilde{n}_A$, satisfy

$$\text{range}((\boldsymbol{J}q_A^j(\boldsymbol{z}))^\top) \subset \ker A(\boldsymbol{z}), \tag{4.5}$$

   where $\boldsymbol{J}q(\boldsymbol{z})$ is the Jacobian matrix of $q(\boldsymbol{z})$.

Assumption 2. Let $A : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ be a matrix-valued function satisfying Assumption 1 and let $\hat{n}_A := n_A - \tilde{n}_A$. There exist real-valued differentiable functions $F_A^j(\cdot) : \mathbb{R}^d \to \mathbb{R}$, $j = 1, \ldots, \hat{n}_A$, on Ω, satisfying

$$\text{span}\{\nabla F_A^j(\boldsymbol{z}) : j = 1, \ldots, \hat{n}_A\} \bigoplus \ker_{\text{inv}} A(\boldsymbol{z}) = \ker A(\boldsymbol{z}). \tag{4.6}$$

The degeneracy conditions of (4.3), if it is interpreted in terms of linear algebra, means that ∇G belongs to the kernel of A. Since $\ker A(\boldsymbol{z})$ depends on $\boldsymbol{z}$, Assumption 1(i) basically allows us to work on the fixed number of basis. This is a typical assumption made in order to make analysis go through, which is also used in [79] dated back in 1970s. Assumption 1(ii) and 2 decompose $\ker A(\boldsymbol{z})$ into two subspaces, one of which satisfies the invariance under differentiation in the sense of (4.5). The other subspace from Assumption 2 provides the key information on how to keep $\nabla G_{\text{NN}}(\boldsymbol{z})$ in the kernel of $A(\boldsymbol{z})$ as a function of $\boldsymbol{z}$. The existence of the functions F_A^j plays a key role in defining the transformation $\mathcal{P}_A$.

With these assumptions, we define a transformation $\mathcal{P}_A$ as follows. From Assumption 1, let $\tilde{Q}_A(\boldsymbol{z}) = [q_A^1(\boldsymbol{z}), \ldots, q_A^{\tilde{n}_A}(\boldsymbol{z})]$. From Assumption 2, let $F_A(\boldsymbol{z}) =$

$[F_A^1(\boldsymbol{z}), \ldots, F_A^{\hat{n}_A}(\boldsymbol{z})]^\top$. Define the transformation $\mathcal{P}_A : \mathbb{R}^d \to \mathbb{R}^{n_A}$ by

$$\mathcal{P}_A(\boldsymbol{z}) = [\tilde{Q}_A^\top(\boldsymbol{z})\boldsymbol{z}; F_A(\boldsymbol{z})] \in \mathbb{R}^{n_A}. \tag{4.7}$$

The first $\tilde{n}_A$ components of $\mathcal{P}_A(\boldsymbol{z})$ is the orthogonal projection coefficients of $\boldsymbol{z}$ onto $\ker_{\text{inv}} A(\boldsymbol{z})$ and the remaining components are the functions from Assumption 2. The output of $\mathcal{P}_A$ is n_A-dimensional vector resulting in a dimension reduction from d to n_A.

We are now in a position to present our neural network for G (either energy E or entropy S). For a neural network $f(\cdot;\theta_G) : \mathbb{R}^d \to \mathbb{R}$ where θ_G is the network parameter, we define

$$G_{\text{NN}}(\boldsymbol{z};\theta_G) = f(\mathcal{P}_A(\boldsymbol{z});\theta_G). \tag{4.8}$$

Since $\nabla(\langle q_A^j(\boldsymbol{z}), \boldsymbol{z}\rangle) = q_A^j(\boldsymbol{z}) + (\mathbf{J}q_A^j(\boldsymbol{z}))^\top \boldsymbol{z}$ and the Jacobian of the transformation (4.7) is

$$(\mathbf{J}\mathcal{P}_A(\boldsymbol{z}))^\top = \left[\nabla(\langle q_A^1(\boldsymbol{z}), \boldsymbol{z}\rangle), \quad \cdots \quad \nabla(\langle q_A^{\tilde{n}_A}(\boldsymbol{z}), \boldsymbol{z}\rangle), \quad \nabla F_A^1(\boldsymbol{z}), \quad \cdots \quad \nabla F_A^{\hat{n}_A}(\boldsymbol{z})\right],$$

it follows from Assumptions 1 and 2 that $\text{range}((\mathbf{J}\mathcal{P}_A(\boldsymbol{z}))^\top) \subset \ker A(\boldsymbol{z})$. It then follows from the multiplicative structure (4.4) that any $G_{\text{NN}}(\cdot)$ of the form (4.8) satisfies $A(\boldsymbol{z})\nabla G_{\text{NN}}(\boldsymbol{z}) = \mathbf{0}$.

We note that in general, the functions from Assumption 2 may not be readily available. However, in Propositions 1 and 2, we show that F_A can be identified by extracting relevant components from the basis of $\ker A$ and then applying indefinite integration.

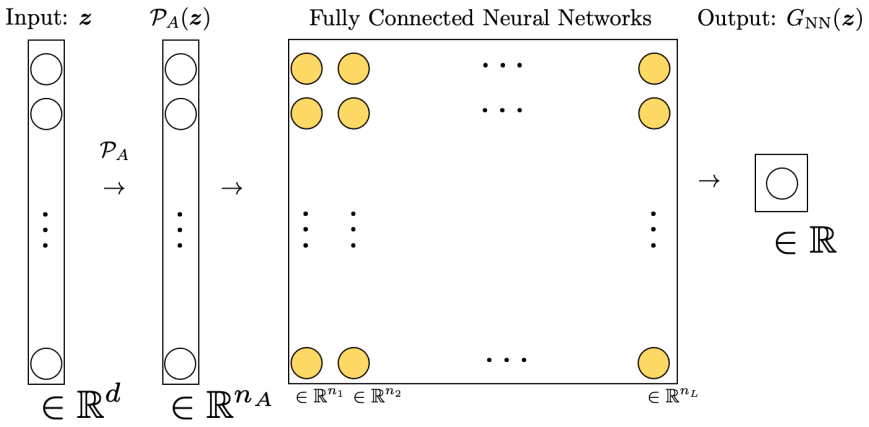


Figure 4.2: **Architecture of GFINNs for Case 1.** We approximate the real-valued function G by G_{NN}, which is parameterized as the composition of the transformation $\mathcal{P}_A$ and a fully connected neural network.The solid dots represent trainable parameters of GFINN in case 1.

The remaining goal is to show the expressivity of the proposed neural network architecture (4.8). In order to show the universal approximation theorem, an appropriate function class should be chosen in the first place on which we show the universality. Since the target function G satisfying the degeneracy condition of (4.3) depends highly on the properties of the kernel of an operator A, a general function class requires some detailed characterizations of $\ker A$. We thus confine ourselves to the function class $\mathcal{F}_A$ characterized by the transformation operator $\mathcal{P}_A$ together with the multiplicative structure (4.4).

Definition 21. Let $A : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ be a matrix-valued function satisfying Assumptions 1 and 2. Let $\mathcal{F}_A$ be the collection of differentiable functions G defined on $\Omega \subset \mathbb{R}^d$ whose gradient satisfies

$$\nabla G(\boldsymbol{z}) = (\boldsymbol{J}\mathcal{P}_A(\boldsymbol{z}))^\top \boldsymbol{c}_A \circ \mathcal{P}_A(\boldsymbol{z}), \qquad \forall \boldsymbol{z} \in \Omega,$$

for some continuous function $\boldsymbol{c}_A : \mathbb{R}^{n_A} \to \mathbb{R}^{n_A}$.

The function class $\mathcal{F}_A$ of Definition 21 depends crucially on A. For example,

as shown in Corollary 4, if $A(\cdot) = A$ is constant, $\mathcal{F}_A$ contains all the continuously differentiable functions G that satisfy $\nabla G \in \ker A$.

Corollary 4. Suppose A is constant. Then, $\mathcal{F}_A$ is the class $C^1(\Omega)$ that consists of all differentiable functions whose gradient lies in $\ker A$ and continuous on Ω.

Proof. Since A is constant, so is Q. Also $n := n_A = \tilde{n}_A$, $\hat{n}_A = 0$, $\mathcal{P}_A(\boldsymbol{z}) = Q^\top \boldsymbol{z}$ and $(\mathbf{J}\mathcal{P}_A(\boldsymbol{z}))^\top = Q$. Let $\mathfrak{G}(\xi) := G(Q\xi)$ where $\xi \in \mathbb{R}^n$. Note that $QQ^\top = I$, thus $Q(Q^\top \boldsymbol{z}) = \boldsymbol{z}$. It then follows from $\nabla G(\boldsymbol{z}) = Q\nabla_\xi G(Q\xi)\big|_{\xi = Q^\top \boldsymbol{z}}$ that $\nabla G(\boldsymbol{z}) = Q\nabla_\xi \mathfrak{G}(\xi)\big|_{\xi = Q^\top \boldsymbol{z}} = Q\nabla_\xi \mathfrak{G} \circ \mathcal{P}_A(\boldsymbol{z})$. By letting $\boldsymbol{c}_A(\xi) = \nabla_\xi \mathfrak{G}(\xi)$, the proof is completed. □

With the target function class being defined, we now show that the proposed neural network G_{NN} defined in (4.8) is universal for the function class $\mathcal{F}_A$.

Theorem 13. Suppose A is a $d \times d$-matrix-valued function on $\Omega \subset \mathbb{R}^d$ satisfying Assumptions 1 and 2. For any $G(\cdot) \in \mathcal{F}_A$ defined as in Definition 21 and a small $\epsilon > 0$, there exists a neural network model $G_{\text{NN}}(\cdot)$ of the form (4.8) such that

$$\sup_{\boldsymbol{z} \in \Omega} \|\nabla G(\boldsymbol{z}) - \nabla G_{\text{NN}}(\boldsymbol{z})\| < \epsilon,$$

and $A(\boldsymbol{z})\frac{\partial G_{\text{NN}}}{\partial \boldsymbol{z}}(\boldsymbol{z}) = \mathbf{0}$ for all $\boldsymbol{z} \in \Omega$.

Proof. Note that $\frac{\partial G_{\text{NN}}}{\partial \boldsymbol{z}}(\boldsymbol{z}) = (\mathbf{J}\mathcal{P}_A(\boldsymbol{z}))^\top \nabla f \circ \mathcal{P}_A(\boldsymbol{z})$. For any $G \in \mathcal{F}_A$, there exists a continuous function $\boldsymbol{c}_A$ such that the gradient of G is expressed as $\nabla G(\boldsymbol{z}) = (\mathbf{J}\mathcal{P}_A(\boldsymbol{z}))^\top \boldsymbol{c}_A \circ \mathcal{P}_A(\boldsymbol{z})$. Since $\mathbf{J}\mathcal{P}_A(\boldsymbol{z})$ is full rank on Ω, by invoking the universal approximation theorem of neural networks (e.g., [126, 177]), for any sufficiently small $\epsilon > 0$, there exists a neural network f satisfying $\sup_{\boldsymbol{z} \in \Omega} \|\nabla f \circ \mathcal{P}_A(\boldsymbol{z}) - \boldsymbol{c}_A \circ \mathcal{P}_A(\boldsymbol{z})\| < \epsilon$, which completes the proof. □

An explicit neural network architecture in terms of width and depth may be given from existing works, however, we simply rely on the well-established universal approximation theorem for neural networks [37, 143, 126, 177].

The considered function class $\mathcal{F}_A$ from Definition 21 may be too restricted to cover more general functions. However, in all the examples of Section 4.4, we show that all the assumptions hold and the underlying energy E and entropy S functions belong to the function classes $\mathcal{F}_M$ and $\mathcal{F}_L$, respectively.

4.3.2 Case 2: L, M, E and S are unknown

We consider only Case 2b where all the quantities (L, M, E, S) are unknown, since Case 2a is easily handled by letting $G_{\text{NN}} := G$.

In Case 2, a scalar function G is modelled by a standard neural network unless it is known a priori. It then suffices to construct a matrix-valued neural network A_{NN} from G_{NN} that satisfies both the symmetry and the degeneracy conditions. Unlike Case 1, we do not need to control the gradient of G_{NN} to be in $\ker A$. Rather, we design A_{NN} to satisfy $\nabla G_{\text{NN}} \in \ker A_{\text{NN}}$. It turns out that one can easily adopt this property into neural network architectures by exploiting skew-symmetric matrices.

Lemma 6. For $j = 1, \dots, K$, let S_j be a skew-symmetric matrix of size $d \times d$. For a differentiable scalar function $g(\cdot) : \mathbb{R}^d \to \mathbb{R}$, let $Q_g(\cdot) : \mathbb{R}^d \to \mathbb{R}^{K \times d}$ be a matrix-valued function whose j-th row is defined to be $(S_j \nabla g(\cdot))^\top$. Then, $Q_g(x) \nabla g(x) = 0$ for all $x \in \mathbb{R}^d$.

Proof. The proof readily follows from the fact that for any $y \in \mathbb{R}^d$, $y^\top S_j y = 0$ since S_j is skew-symmetric. □

For a neural network G_{NN}, let $Q_{G_{\text{NN}}}$ be the matrix function defined through K skew-symmetric matrices as in Lemma 6, which are trainable parameters. Motivated by the spectral decomposition of either skew-symmetric or symmetric positive semi-definite matrix, we propose to model A by

$$A_{\text{NN}}(\boldsymbol{z}) := (Q_{G_{\text{NN}}}(\boldsymbol{z}))^\top B^A_{\text{NN}}(\boldsymbol{z}) Q_{G_{\text{NN}}}(\boldsymbol{z}), \tag{4.9}$$

where B^A_{NN} is skew-symmetric if $A = L$ and is symmetric positive semi-definite if $A = M$ that is modelled by another neural network different from G_{NN}. In particular, we use two triangular matrix-valued neural networks $T_L(\boldsymbol{z})$ and $T_M(\boldsymbol{z})$ and set

$$B^L_{\text{NN}}(\boldsymbol{z}) := (T_L(\boldsymbol{z}))^\top - T_L(\boldsymbol{z}), \qquad B^M_{\text{NN}}(\boldsymbol{z}) := (T_M(\boldsymbol{z}))^\top T_M(\boldsymbol{z}). \tag{4.10}$$

By construction, the symmetry and degeneracy conditions of (4.3) are automatically satisfied.

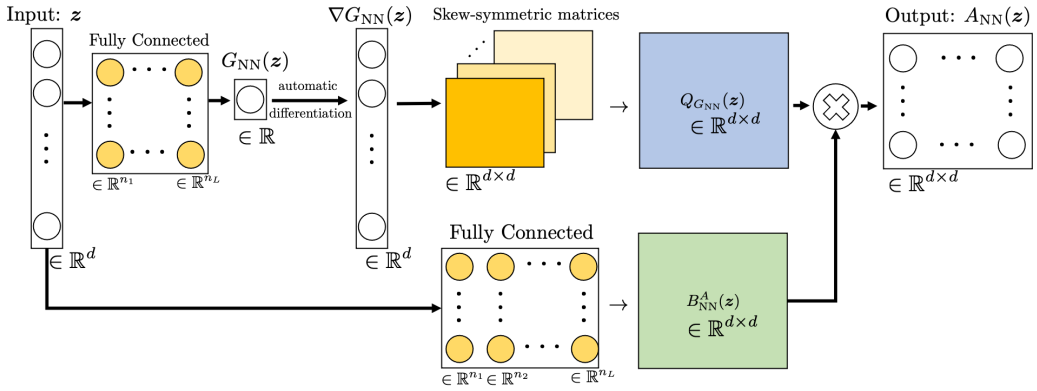


Figure 4.3: **Architecture of GFINNs for Case 2.** We propose a new architecture to parameterize the real-valued function G and matrix-valued function A simultaneously. Three major components of the architecture include the fully connected neural network G_{NN}, skew-symmetric matrices S_j and structured (skew-symmetric or symmetric positive semi-definite) neural networks B^A_{NN}. Their compositions give the output A_{NN}. The trainable parameters are highlighted.

We note that if $A(\boldsymbol{z})$ is either skew-symmetric or symmetric positive semi-definite, the spectral decomposition reads $A(\boldsymbol{z}) = Q(\boldsymbol{z})^\top \Lambda(\boldsymbol{z}) Q(\boldsymbol{z})$, where $Q(\boldsymbol{z})$ is orthogonal

and $\Lambda(\boldsymbol{z})$ is either skew-symmetric or diagonal. The network architecture of (4.9) has a similar structure of that of the spectral decomposition, yet, neither $Q_{G_{\text{NN}}}(\boldsymbol{z})$ is orthogonal nor $B^A_{\text{NN}}(\boldsymbol{z})$ is the eigenvalue matrix. Since $Q_{G_{\text{NN}}}(\boldsymbol{z})$ is a matrix of size $K \times d$, the rank of $A_{\text{NN}}(\boldsymbol{z})$ is at most $\min\{K, d\}$. Hence, K is assumed to be greater than or equal to the rank of $A(\boldsymbol{z})$.

Owing to the universal approximation theorem [37, 143] of neural networks, we show that the proposed neural network A_{NN} of (4.9) is sufficiently expressive enough to approximate the underlying target function A under some mild conditions.

Theorem 14. Suppose $\nabla G_{\text{NN}} := \nabla G$ is continuous on a compact set $\Omega \subset \mathbb{R}^d$, and a component of ∇G has nonzero values in Ω. Let A be either a skew-symmetric or symmetric positive semi-definite matrix-valued continuous function satisfying $A(\boldsymbol{z})\nabla G(\boldsymbol{z}) = 0$ for all $z \in \Omega$. For any $\epsilon > 0$, there exists a neural network model A_{NN} of the form (4.9) such that $\sup_{\boldsymbol{z}\in\Omega} \|A - A_{\text{NN}}\| < \epsilon$ and $A_{\text{NN}}(\boldsymbol{z})\nabla G(\boldsymbol{z}) = \mathbf{0}$ in Ω.

Proof. Without loss of generality, let $(\nabla G(\boldsymbol{z}))_1 \neq 0$ for all $z \in \Omega$. Since $A(\boldsymbol{z})\nabla G(\boldsymbol{z}) = 0$, the row space, $\text{row}(A(\boldsymbol{z}))$, of $A(\boldsymbol{z})$ is spanned by at most $d-1$ independent basis. For $k = 1, \ldots, d-1$, let P_k be a skew-symmetric matrix of size d such that $(P_k)_{1,k+1} = 1$, $(P_k)_{k+1,1} = -1$ and $(P_k)_{ij} = 0$ otherwise. Let $\tilde{q}_k(\boldsymbol{z}) := P_k \nabla G(\boldsymbol{z})$. Note that $(\tilde{q}_k(\boldsymbol{z}))_1 = (\nabla G(\boldsymbol{z}))_{k+1}$, $(\tilde{q}_k(\boldsymbol{z}))_{k+1} = -(\nabla G(\boldsymbol{z}))_1$, and $(\tilde{q}_k(\boldsymbol{z}))_i = 0$ for all other i.

Claim 1. $\tilde{q}_k(\boldsymbol{z})$, $1 \leq k < d$, are linearly independent and

$$\textit{row}(A(\boldsymbol{z})) \subseteq \textit{span}\{\tilde{q}_k(\boldsymbol{z}) : 1 \leq k < d\}.$$

Proof of Claim. Assuming $\sum_{k=1}^{d-1} c_k \tilde{q}_k(\boldsymbol{z}) = 0$, it suffices to show $c_k = 0$ for all k.

Observe that

$$\sum_{k=1}^{d-1} c_k \tilde{q}_k(\boldsymbol{z}) = \left[\sum_{k=1}^{d-1} c_k (\nabla G(\boldsymbol{z}))_{k+1}, \quad -c_1(\nabla G(\boldsymbol{z}))_1, \quad \cdots \quad -c_{d-1}(\nabla G(\boldsymbol{z}))_1\right]^\top.$$

Since $(\nabla G(\boldsymbol{z}))_1 \neq 0$, $c_1 = \cdots = c_{d-1} = 0$, which proves the linearly independence. Note also that since P_k is skew-symmetric, $\langle \tilde{q}_k(\boldsymbol{z}), \nabla G(\boldsymbol{z})\rangle = \langle P_k \nabla G(\boldsymbol{z}), \nabla G(\boldsymbol{z})\rangle = 0$ for all k. Since $\mathrm{row}(A(\boldsymbol{z}))$ belongs to the orthogonal complement space of $\mathrm{span}\{\nabla G(\boldsymbol{z})\}$, the second claim is followed from the relationship

$$\mathrm{row}(A(\boldsymbol{z})) \subseteq (\mathrm{span}\{\nabla G(\boldsymbol{z})\})^\perp = \mathrm{span}\{\tilde{q}_k(\boldsymbol{z}) : 1 \leq k < d\},$$

where the equality holds because the two spaces have the same rank. □

Note that $A(\boldsymbol{z})$ is either skew-symmetric or symmetric positive semi-definite of rank r with $2 \leq r < d$. Thus, $\mathrm{row}(A(\boldsymbol{z})) = \mathrm{col}(A(\boldsymbol{z}))$. We shall consider only the case when $A(\boldsymbol{z})$ is skew-symmetric. The other case can be proved similarly. By the spectral decomposition, we have $A(\boldsymbol{z}) = Q(\boldsymbol{z})\Lambda(\boldsymbol{z})Q^\top(\boldsymbol{z})$, where $Q(\boldsymbol{z})$ is a matrix of size $d \times r$ such that $Q^\top(\boldsymbol{z})Q(\boldsymbol{z}) = I \in \mathbb{R}^{r\times r}$ and $\Lambda(\boldsymbol{z})$ is skew-symmetric of size $r \times r$ defined by

$$(\Lambda(\boldsymbol{z}))_{ij} = \begin{cases} \lambda_i(\boldsymbol{z}) & \text{if } j = i+1, \\ -\lambda_i(\boldsymbol{z}) & \text{if } j = i-1, \\ 0 & \text{otherwise,} \end{cases}$$

where $\lambda_i(\boldsymbol{z}) > 0$. Since the columns of $Q(\boldsymbol{z})$ are orthonormal basis for $\mathrm{col}(A)$, and from Claim 1, $Q(\boldsymbol{z})$ can be represented by $\{\tilde{q}_k(\boldsymbol{z})\}$. That is, there exists a matrix $R(\boldsymbol{z}) \in \mathbb{R}^{(d-1)\times r}$ such that $Q(\boldsymbol{z}) = \tilde{Q}(\boldsymbol{z})R(\boldsymbol{z})$, where $\tilde{Q}(\boldsymbol{z}) = [\tilde{q}_1(\boldsymbol{z}), \cdots, \tilde{q}_{d-1}(\boldsymbol{z})]$. Since $Q^\top(\boldsymbol{z})\tilde{Q}(\boldsymbol{z})$ is full rank and $r \leq d-1$, a solution to $Q^\top(\boldsymbol{z})\tilde{Q}(\boldsymbol{z})R(\boldsymbol{z}) = I$ always exists.

Since A is continuous on $\Omega \subset \mathbb{R}^d$, so does $\tilde{\Lambda}(\cdot) := R(\cdot)\Lambda(\cdot)R^\top(\cdot)$. From the universal approximation theorem of neural networks, there exists a skew-symmetric matrix-valued neural network $B^A_{\mathrm{NN}}(\cdot)$ such that

$$\|(B^A_{\mathrm{NN}}(\cdot))_{ij} - (\tilde{\Lambda}(\cdot))_{ij}\|_{C^0(\Omega)} < \frac{\epsilon}{C(d-1)r}, \qquad \forall 1 \le i < j < d, \tag{4.11}$$

where $C = \sup_{\boldsymbol{z}\in\Omega} \|\tilde{Q}(\boldsymbol{z})\|^2$. Note that C is a constant that depends only on $\nabla G(\boldsymbol{z})$ and Ω. Let $A_{\mathrm{NN}}(\cdot) := \tilde{Q}(\cdot)B^A_{\mathrm{NN}}(\cdot)\tilde{Q}^\top(\cdot)$. Then,

$$\begin{aligned}
\|A(\boldsymbol{z}) - A_{\mathrm{NN}}(\boldsymbol{z})\| &= \|Q(\boldsymbol{z})\Lambda(\boldsymbol{z})Q^\top(\boldsymbol{z}) - \tilde{Q}(\boldsymbol{z})B^A_{\mathrm{NN}}(\boldsymbol{z})\tilde{Q}^\top(\boldsymbol{z})\| \\
&= \|\tilde{Q}(\boldsymbol{z})R(\boldsymbol{z})\Lambda(\boldsymbol{z})R^\top(\boldsymbol{z})\tilde{Q}^\top(\boldsymbol{z}) - \tilde{Q}(\boldsymbol{z})B^A_{\mathrm{NN}}(\boldsymbol{z})\tilde{Q}^\top(\boldsymbol{z})\| \\
&\le \|\tilde{Q}(\boldsymbol{z})\|^2\|R(\boldsymbol{z})\Lambda(\boldsymbol{z})R^\top(\boldsymbol{z}) - B^A_{\mathrm{NN}}(\boldsymbol{z})\|.
\end{aligned}$$

Since $\|\cdot\| \le \|\cdot\|_F$, where $\|\cdot\|_F$ is the Frobenius norm, it follows from (4.11) that $\|A(\boldsymbol{z}) - A_{\mathrm{NN}}(\boldsymbol{z})\| \le \epsilon$ for all $\boldsymbol{z} \in \Omega$, which completes the proof. □

Assuming Case 2b, the GENERIC formalism informed neural networks (GFINNs) comprise of four neural networks: $E_{\mathrm{NN}}, S_{\mathrm{NN}}, B^L_{\mathrm{NN}}, B^M_{\mathrm{NN}}$ together with sets of trainable parameters forming skew-symmetric matrices used in $Q_{E_{\mathrm{NN}}}$ and $Q_{S_{\mathrm{NN}}}$. Apart from parameters for the four neural networks, the number of parameters from the skew-symmetric matrices is $2Kd(d-1)$. When the dimension d is large, the number of trainable parameters grows $\mathcal{O}(Kd^2)$, which may cause some computational challenges. If this is the case, sparse parameterization can be applied to reduce the number of parameters. For example, each skew-symmetric matrix could be sparsely parameterized by only s nonzero parameters, which reduces the number of trainable parameters from $\mathcal{O}(Kd^2)$ to $\mathcal{O}(Ks)$. A similar sparse parameterization can be applied in modelling B^A_{NN} as well.

In summary, GFINNs consist of four components, E_{NN}, S_{NN}, L_{NN}, M_{NN} that form two modules; E-M module (E_{NN}, M_{NN}) and L-S module (L_{NN},S_{NN}). When A is known, we model G using a neural network G_{NN} of the form (4.8). When G is known, we model A using a neural network A_{NN} of the form (4.9). When no physics is known, we model G using a standard neural network G_{NN} and then model A using a neural network A_{NN} of the form (4.9) with G_{NN}. As a result, we obtain a general framework of designing neural networks for the GENERIC formalism with the flexibility of incorporating available physics, thanks to the component-wise network modelling.

4.4 Numerical Examples

We demonstrate the performance of GFINNs on three benchmark problems. To compare against other methods, we also report the results obtained by GNODEs [122], SPNNs [87], and SDENets [50]. Implementation details can be found in Appendix 4.5. The code for the numerical examples are openly available on GitHub at https://github.com/zzhang222/gfinn_gc. We note that SPNNs and GNODEs can only be applied in case 1 and case 2, respectively, while GFINNs cover all the cases.

Nonuniqueness. By the multiplicative structure, the GENERIC formalism allows multiple modules resulting in the same dynamics. That is, given (E, S, L, M), there are infinitely many $(\tilde{E}, \tilde{S}, \tilde{L}, \tilde{M})$ satisfying $\tilde{L}(\boldsymbol{z})\frac{\partial \tilde{E}}{\partial \boldsymbol{z}}(\boldsymbol{z})+\tilde{M}(\boldsymbol{z})\frac{\partial \tilde{S}}{\partial \boldsymbol{z}}(\boldsymbol{z}) = L(\boldsymbol{z})\frac{\partial E}{\partial \boldsymbol{z}}(\boldsymbol{z})+M(\boldsymbol{z})\frac{\partial S}{\partial \boldsymbol{z}}(\boldsymbol{z})$. As a matter of fact, if we let $\tilde{G}$ be an affine transformation of G with a slope coefficient a, by letting $\tilde{A} := a^{-1}A$, we obtain new modules resulting in the same dynamics. Because of the nonuniqueness, the inferred quantities E_{NN} and S_{NN} may look different from the target quantities. We therefore calibrate the inferred

quantities to make them look similar to the ground truth values. The calibration is done by finding aforementioned affine transformations using some target values. We apply the calibration only for the visualization purpose to demonstrate the discovery of the physical quantities by GFINNs.

GENERIC formalism under fluctuations. Fluctuations can be included in the GENERIC formalism [82, 149], resulting in a stochastic differential equation (SDE) of the form

$$\begin{aligned} & d\boldsymbol{z} = \mu(\boldsymbol{z})dt + \sigma(\boldsymbol{z})d\boldsymbol{W}_t, \\ \text{subject to} \quad & L(\boldsymbol{z})\frac{\partial S}{\partial \boldsymbol{z}}(\boldsymbol{z}) = \sigma(\boldsymbol{z})\frac{\partial E}{\partial \boldsymbol{z}}(\boldsymbol{z}) = \boldsymbol{0}, \\ & L(\boldsymbol{z}) \text{ is skew-symmetric, i.e., } L(\boldsymbol{z}) = -L(\boldsymbol{z})^\top, \\ & \sigma(\boldsymbol{z})\sigma(\boldsymbol{z})^\top = 2k_B M(\boldsymbol{z}), \end{aligned} \tag{4.12}$$

where $\mu(\boldsymbol{z}) := L(\boldsymbol{z})\frac{\partial E}{\partial \boldsymbol{z}}(\boldsymbol{z}) + M(\boldsymbol{z})\frac{\partial S}{\partial \boldsymbol{z}}(\boldsymbol{z}) + k_B\frac{\partial}{\partial \boldsymbol{z}} \cdot M(\boldsymbol{z})$, $\boldsymbol{W}_t$ is a multicomponent Wiener process and k_B is the Bolzmann constant, which controls the magnitude of fluctuation. When the fluctuations are eliminated by letting $k_B \to 0$, we recover (4.3) from (4.12). Here $\frac{\partial}{\partial \boldsymbol{z}} \cdot M$ represents the divergence of M as a tensor field, i.e., $\frac{\partial}{\partial \boldsymbol{z}} \cdot M = \sum_{i=1}^d \sum_{k=1}^d \frac{\partial M_{ik}}{\partial \boldsymbol{z}_k}\boldsymbol{e}_i$ where $\boldsymbol{e}_i$'s are the standard basis vectors in $\mathbb{R}^d$. The consistency condition of (4.12) implies the conservation of energy and fluctuation-dissipation theorem.

In the stochastic setting, the goal is to infer the drift and diffusion terms μ, σ from observed data. Let $Z(t, \omega; Z_0)$ be the solution to the SDE (4.12), where ω denotes that $Z(t, \omega; Z_0)$ is a random variable and possesses the initial condition $Z(t_0, \omega; Z_0) = Z_0$ with probability one. The data are then referred to N_{traj} sample paths of the solution to (4.12), each of which has different initial states, $\{Z_0^{(k)}\}$, sampled from a probability distribution. The k-th sample path is then written as

$\{t_j, \boldsymbol{z}_j^{(k)}\}_{j=0}^T$ where $\boldsymbol{z}_j^{(k)} := Z(t_j, \omega^{(k)}; Z_0^{(k)})$ and $\omega^{(k)}$ is a realization of outcome.

GFINNs for the SDE (4.12) consist of the components $A_{\text{NN}}, G_{\text{NN}}, Q_{E_{\text{NN}}}, T_M$ from (4.9) and (4.10), which further construct μ_{NN} and σ_{NN} as follows:

$$\begin{aligned} \mu_{\text{NN}}(\boldsymbol{z}) &:= L_{\text{NN}}(\boldsymbol{z})\frac{\partial E_{\text{NN}}}{\partial \boldsymbol{z}}(\boldsymbol{z}) + M_{\text{NN}}(\boldsymbol{z})\frac{\partial S_{\text{NN}}}{\partial \boldsymbol{z}}(\boldsymbol{z}) + k_B\frac{\partial}{\partial \boldsymbol{z}} \cdot M_{\text{NN}}(\boldsymbol{z}), \\ \sigma_{\text{NN}}(\boldsymbol{z}) &:= \sqrt{2k_B}(T_M(\boldsymbol{z})Q_{E_{\text{NN}}}(\boldsymbol{z}))^\top. \end{aligned} \tag{4.13}$$

Both $\mu_{\text{NN}}(\boldsymbol{z})$ and σ_{NN} are naturally defined thanks to the spectral structure of A_{NN}. Since each component obeys the required conditions of (4.3), GFINNs (4.13) satisfy the consistency conditions of (4.12).

Loss function. In the deterministic examples, the loss function is set to the mean squared error (MSE) defined in (4.2) together with the Runge-Kutta second/third order integrator [118].

In the stochastic example, the loss function is set to the negative log-likelihood function (4.14) together with the Euler-Maruyama integrator [111]. The same loss function is also used in [50, 174], which is defined by

$$\mathcal{L}(\theta) = -\frac{1}{N_{\text{traj}}}\sum_{k=1}^{N_{\text{traj}}}\frac{1}{T}\sum_{j=1}^{T}\log p(\boldsymbol{z}_j^{(k)}|\boldsymbol{z}_{j-1}^{(k)}, \theta), \tag{4.14}$$

where $p(\boldsymbol{z}_j^{(k)}|\boldsymbol{z}_{j-1}^{(k)}, \theta)$ is the probability density function of multivariate normal distribution with mean $\Delta t_j \mu_{\text{NN}}(\boldsymbol{z}_{j-1}^{(k)})$ and covariance matrix $2k_B\Delta t_j M_{\text{NN}}(\boldsymbol{z}_{j-1}^{(k)})$ evaluated at $\boldsymbol{z}_j^{(k)} - \boldsymbol{z}_{j-1}^{(k)}$. Here $\Delta t_j := t_j - t_{j-1}$.

Evaluation metric. The performance quality of learned dynamics is measured by a closeness between unseen ground truth trajectories that are not used in training, and trajectories of inferred dynamics. This is often referred to as generalization or

test error.

Let N_{test} be the number of unseen test trajectories. For $k = 1, \ldots, N_{\text{test}}$, let $\boldsymbol{Z}^{(k)} \in \mathbb{R}^{T \times d}$ be the matrix representing the k-th *unseen* trajectory of ground-truth, whose j-th row, denoted by $\boldsymbol{Z}_j^{(k)}$, is the state at time t_j. Similarly, let $\tilde{\boldsymbol{Z}}^{(k)} \in \mathbb{R}^{T \times d}$ be the k-trajectory matrix of learned dynamics whose initial state is the same as the one of $\boldsymbol{Z}^{(k)}$.

In the deterministic case, the metric we use for closeness is the mean squared error (MSE):

$$\text{MSE}(t_j) = \frac{1}{N_{\text{test}}} \sum_{k=1}^{N_{\text{test}}} \frac{1}{d} \sum_{l=1}^{d} (\boldsymbol{Z}_{jl}^{(k)} - \tilde{\boldsymbol{Z}}_{jl}^{(k)})^2. \tag{4.15}$$

In the stochastic case, the metric we use for closeness is the squared sliced Wasserstein-2 distance [48], which is defined through random projections. Let $\{\boldsymbol{u}_m\}_{m=1}^{M}$ be a set of M vectors randomly uniformly sampled from the unit hypersphere $\mathbb{S}^{d-1}$, where we set $M = 100$ for implementation. For each j and m, $\langle \boldsymbol{Z}_j^{(k)}, \boldsymbol{u}_m \rangle$ and $\langle \tilde{\boldsymbol{Z}}_j^{(k)}, \boldsymbol{u}_m \rangle$ are always assumed to be sorted with respect to the index k. The squared sliced Wasserstein-2 distance (SW) is then defined by

$$\text{SW}(t_j) = \frac{1}{N_{\text{test}}} \sum_{k=1}^{N_{\text{test}}} \frac{1}{M} \sum_{m=1}^{M} |\langle \boldsymbol{Z}_j^{(k)} - \tilde{\boldsymbol{Z}}_j^{(k)}, \boldsymbol{u}_m \rangle|^2.$$

4.4.1 Two gas containers exchanging heat and volume

We consider the gas container example from [176]. Two gas containers are allowed to exchange heat and volume with a wall in the middle. The state variable is $\boldsymbol{z} = (q, p, S_1, S_2) \in \mathbb{R}^4$, where q, p represent the position and momentum of the moving wall, and S_1, S_2 represent the entropy of the gases in two containers. The energy of

the whole system is $E(\boldsymbol{z}) = \frac{p^2}{2m} + E_1 + E_2$, where $E_i = (\frac{e^{\frac{S_i}{Nk_B}}}{\hat{c}V_i})^{\frac{2}{3}}$, $V_1 = q$, $V_2 = 2-q$ and $\hat{c} = \left(\frac{4\pi m}{3h^2 N}\right)^{\frac{3}{2}} \frac{e^{\frac{5}{2}}}{N}$, which follows from the Sackur–Tetrode equation [175] for ideal gases. m is the mass of the wall, N is the number of gas particles, h is the Planck constant and k_B is the Boltzmann constant. We fix the units such that $m = Nk_B = \hat{c} = 1$. The entropy of the system is $S(\boldsymbol{z}) = S_1 + S_2$. The evolution equation is described by a system of ordinary differential equations (ODEs):

$$\begin{pmatrix} \dot{q} \\ \dot{p} \\ \dot{S}_1 \\ \dot{S}_2 \end{pmatrix} = \begin{pmatrix} \frac{p}{m} \\ \frac{2}{3}(\frac{E_1}{q} - \frac{E_2}{2-q}) \\ \frac{\alpha}{T_1}(\frac{1}{T_1} - \frac{1}{T_2}) \\ -\frac{\alpha}{T_2}(\frac{1}{T_1} - \frac{1}{T_2}) \end{pmatrix} = L\frac{\partial E(\boldsymbol{z})}{\partial \boldsymbol{z}} + M(\boldsymbol{z})\frac{\partial S(\boldsymbol{z})}{\partial \boldsymbol{z}}, \tag{4.16}$$

where $L = \begin{bmatrix} \boldsymbol{S} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$, $M = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T} \end{bmatrix}$, $\boldsymbol{S} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, $\boldsymbol{T} = \alpha \begin{bmatrix} T_1^{-2} & -(T_1T_2)^{-1} \\ -(T_1T_2)^{-1} & T_2^{-2} \end{bmatrix}$, $T_i = \frac{\partial E_i}{\partial S_i}$, $\boldsymbol{0}$ is the zero matrix of size 2×2 and α is the parameter determining the strength of heat exchange which we set to 10.

The following proposition shows that the governing equation (4.16) of the gas container problem satisfies all the assumptions of Section 4.3.

Proposition 1. Let $M(\boldsymbol{z})$ be the matrix defined in (4.16). Let

$$F_M(\boldsymbol{z}) = e^{CS_1} q^{-\frac{2}{3}} + e^{CS_2}(2-q)^{-\frac{2}{3}},$$

where $C = \frac{2}{3Nk_B}$. Then, $\ker M(\boldsymbol{z}) = \operatorname{span}\{e_1, e_2, \nabla F_M(\boldsymbol{z})\}$, where

$$\nabla F_M(\boldsymbol{z}) = (T_3, 0, T_1, T_2)^\top, \qquad T_3 = -\frac{2}{3}e^{\frac{2}{3}CS_1} q^{-\frac{5}{3}} + \frac{2}{3}e^{\frac{2}{3}CS_2}(2-q)^{-\frac{5}{3}}.$$

Let $\hat{q}(\boldsymbol{z}) = (0, 0, \bar{T}_1, \bar{T}_2)^\top$ where $\bar{T}_i = \frac{T_i}{\sqrt{T_1^2 + T_2^2}}$. Then, $\{e_1, e_2, \hat{q}(\boldsymbol{z})\}$ is an orthonormal

basis for $\ker M(\boldsymbol{z})$ and $\text{range}((\boldsymbol{J}\hat{q}(\boldsymbol{z}))^\top) \not\subset \ker M(\boldsymbol{z})$. Furthermore, $E \in \mathcal{F}_M$ as

$$\nabla E(\boldsymbol{z}) = (\boldsymbol{J}\mathcal{P}_M(\boldsymbol{z}))^\top \boldsymbol{c}_M \circ \mathcal{P}_M(\boldsymbol{z}),$$

where $\mathcal{P}_M(\boldsymbol{z}) = (q, p, F_M(\boldsymbol{z}))^\top$, $\boldsymbol{c}_M(\xi_1, \xi_2, \xi_3) = (1, \xi_2/m, 1)^\top$, and $(\boldsymbol{J}\mathcal{P}_M(\boldsymbol{z}))^\top = [e_1, e_2, \nabla F_M(\boldsymbol{z})]$.

Proof. The proof directly follows from a straight forward calculation. □

We use 80 trajectories starting from $t_0 = 0$ to $t_T = 8$ with $\Delta t_j := t_j - t_{j-1} = 0.02$ ($T = 400$) as training data and use another 20 trajectories as test data. The initial conditions of both training and testing trajectories are uniformly sampled from $[0.2, 1.8] \times [-1, 1] \times [1, 3] \times [1, 3]$. Since neural networks may be sensitive to how parameters are initialized, we run ten independent simulations and report some ensembles out of it. As an effort to make a fair comparison, neural networks used for each method have a roughly similar number of parameters. The detailed architectures are summarized in Table 4.2 of Appendix 4.5.

In the top row of Figure 4.4, we plot one of 20 test trajectories as dashed-lines, together with the corresponding trajectory of the learned dynamics by GFINNs as symbols; the circle ($\circ$), the inverted triangle ($\triangledown$) and the cross ($\times$) marks correspond to case 1, case 2a and case 2b, respectively. Here, GFINNs are the one with the smallest MSE summed over time among 10 simulations. Since the state variable lies in $\mathbb{R}^4$, the q, p and $S_1 + S_2$ trajectories are reported. We clearly see that in case 1 and case 2a, the trajectories of GFINNs are indistinguishable to the ground truth trajectory, while in case 2b, they start to deviate from the truth trajectory as time increases. This is expected as some underlying physics in terms of the GENERIC formalism is known in case 1 and case 2a, while no physics is known in case 2b.

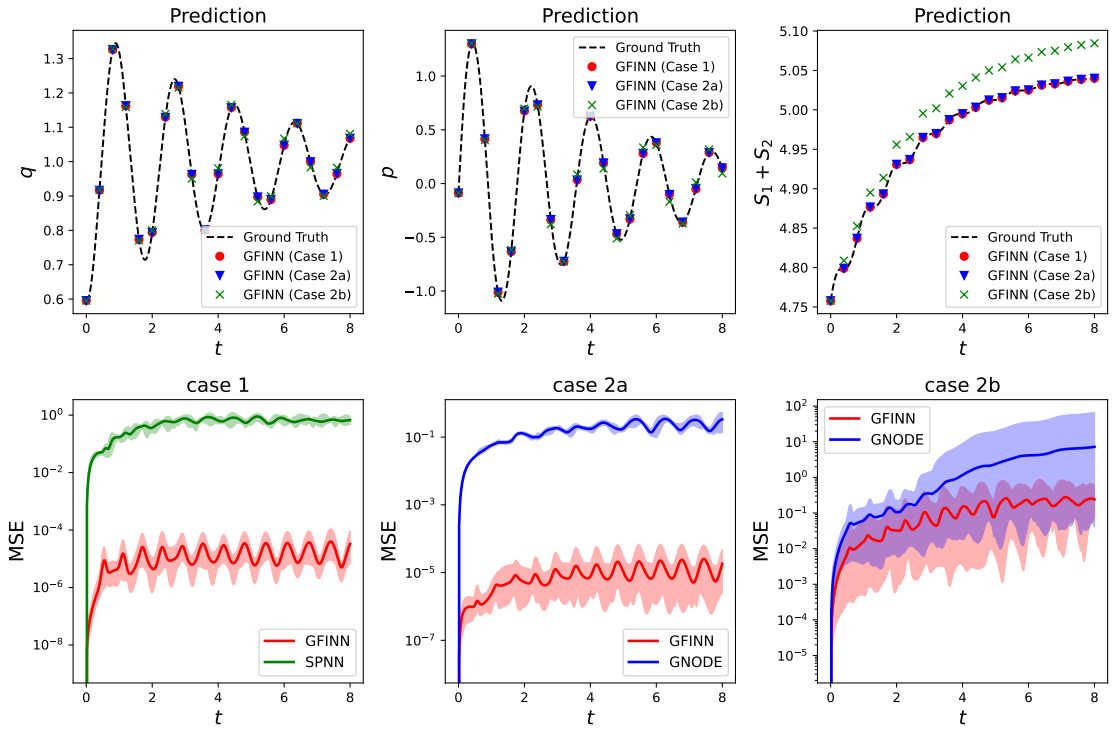


Figure 4.4: **Results for the gas container example.** (**Top-left and Top-middle**) Predicted position q and momentum p for the wall. (**Top-right**) Predicted entropy $S_1 + S_2$ of the air in two gas containers. (**Bottom-left**) The prediction MSE of GFINN and SPNN in case 1. (**Bottom-middle and bottom-right**) The prediction MSE of GFINN and GNODE in case 2a and case 2b. The results are obtained by taking the mean of 10 simulations, while the upper and lower boundary for shaded region represents the highest and lowest error in 10 simulations.

To compare against other methods, in the bottom row of Figure 4.4, we report the means of the MSE (4.15) of GFINNs, GNODEs and SPNNs with respect to time. Each shaded region represents the range between the maximum and the minimum of the MSEs from ten simulations. We clearly observe that the mean of the MSEs by GFINNs is much lower than those by SPNNs and GNODEs in all the cases. In both case 1 and case 2a, all the 10 MSEs of GFINNs are significantly lower than those of the other comparisons. This again demonstrates that by incorporating physical knowledge into neural networks, GFINNs can achieve much higher predictive accuracy. In case 2b, the mean MSE of GFINNs is at least one order magnitude smaller than the one of GNODEs in almost all times. These results indicate that to

achieve good performance, it is not enough to just enforce the GENERIC conditions, but a sufficient expressivity is also required to capture the underlying dynamics.

In Figure 4.5, we plot the contours of energy and entropy functions from both the ground truth and GFINNs in all the cases. Due to the nonuniquness of the GENERIC formalism, a proper calibration is applied. We see that the calibrated contours of both the energy and entropy by GFINNs are indistinguishable to those by the ground truth. This demonstrates the discovery of the energy and entropy by GFINNs from data.

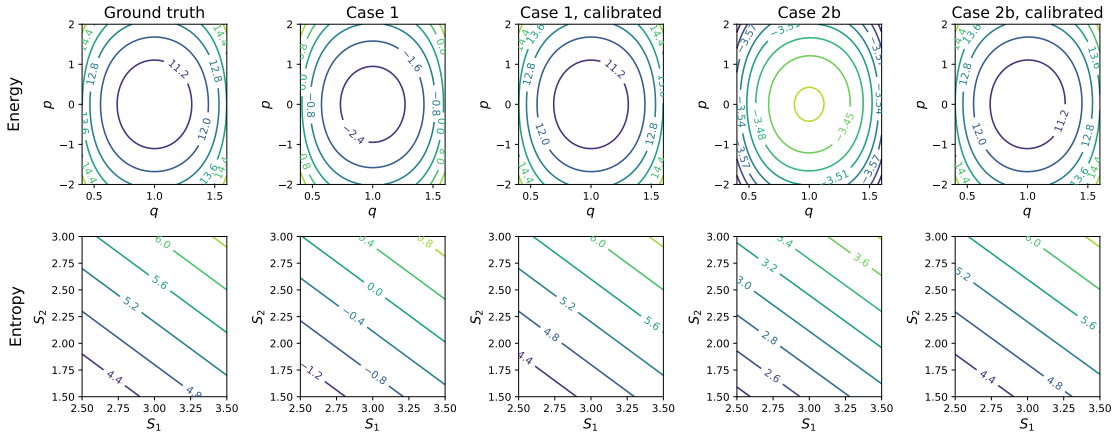


Figure 4.5: **Learned energy E and entropy S of the gas container example by GFINN.** In both case 1 and case 2b, the GFINNs can recover the correct energy and entropy of the system when the undetermined scaling and translation factor is calibrated with the ground truth. We show the energy surface at the hyperplane $\{z : S_1 = 2.5, S_2 = 2.5\}$ as a function of p, q and the entropy surface at $\{z : p = 0, q = 1\}$ as a function of S_1 and S_2.

4.4.2 Thermoelastic double pendulum

We consider the two-dimensional finite thermoelastic double pendulum example [166]. The state variable is $\boldsymbol{z} = (\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{p}_1, \boldsymbol{p}_2, S_1, S_2) \in \mathbb{R}^{10}$, where $\boldsymbol{q}_i, \boldsymbol{p}_i \in \mathbb{R}^2$ represent the position, momentum of the i-th mass, and S_i represents the entropy of

the i-th spring. We denote the length of two springs as $\lambda_1 = \|\boldsymbol{q}_1\|$, $\lambda_2 = \|\boldsymbol{q}_2 - \boldsymbol{q}_1\|$. The total energy of the system is $E(\boldsymbol{z}) = \frac{\|\boldsymbol{p}_1\|^2}{2} + \frac{\|\boldsymbol{p}_2\|^2}{2} + E_1 + E_2$, where E_1 and E_2 are the internal energy of both springs, defined by $E_i = \frac{1}{2}(\log \lambda_i)^2 + \log \lambda_i + e^{S_i - \log \lambda_i} - 1$. The entropy of the system is $S(\boldsymbol{z}) = S_1 + S_2$. The evolution equation for the problem is then given by a system of ODEs:

$$\begin{pmatrix} \dot{\boldsymbol{q}}_1 \\ \dot{\boldsymbol{q}}_2 \\ \dot{\boldsymbol{p}}_1 \\ \dot{\boldsymbol{p}}_2 \\ \dot{S}_1 \\ \dot{S}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ -\frac{\partial}{\partial \boldsymbol{q}_1}(E_1 + E_2) \\ -\frac{\partial}{\partial \boldsymbol{q}_2}(E_1 + E_2) \\ T_1^{-1} T_2 - 1 \\ T_1 T_2^{-1} - 1 \end{pmatrix} = L \frac{\partial E(\boldsymbol{z})}{\partial \boldsymbol{z}} + M(\boldsymbol{z}) \frac{\partial S(\boldsymbol{z})}{\partial \boldsymbol{z}}, \tag{4.17}$$

where $T_i = \frac{\partial E_i}{\partial S_i}$, $\mathbf{0}$ and $\mathbf{1}$ are 2×2 matrices whose elements are all 0 and 1, respectively, $\mathbf{0}_{m_1 \times m_2}$ is a matrix of size $m_1 \times m_2$ whose elements are all 0,

$$L = \begin{pmatrix} \mathbf{0}_{4\times 4} & \boldsymbol{S} \\ -\boldsymbol{S}^\top & \mathbf{0}_{6\times 6} \end{pmatrix}, M(\boldsymbol{z}) = \begin{pmatrix} \mathbf{0}_{8\times 8} & \mathbf{0}_{8\times 2} \\ \mathbf{0}_{2\times 8} & \boldsymbol{T}(\boldsymbol{z}) \end{pmatrix}, \boldsymbol{S} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}, \boldsymbol{T} = \begin{pmatrix} \frac{T_2}{T_1} & -1 \\ -1 & \frac{T_1}{T_2} \end{pmatrix}. \tag{4.18}$$

The governing equation (4.17) for the problem again satisfies all the assumptions in Section 4.3, which is shown in the following proposition.

Proposition 2. Let $M(\boldsymbol{z})$ be the matrix defined in (4.18). Let $F_M(\boldsymbol{z}) = -\|\boldsymbol{q}_1\| e^{S_1} - \|\boldsymbol{q}_2 - \boldsymbol{q}_1\| e^{S_2}$. Then, $\ker M(\boldsymbol{z}) = \operatorname{span}\{e_1, \ldots, e_8, \nabla F_M(\boldsymbol{z})\}$ and $\nabla F_M(\boldsymbol{z}) = (T_3; T_4; \mathbf{0}_{4\times 1}; T_1; T_2)$ where

$$T_3 := -\frac{\boldsymbol{q}_1}{\|\boldsymbol{q}_1\|} e^{S_1} + \frac{\boldsymbol{q}_2 - \boldsymbol{q}_1}{\|\boldsymbol{q}_2 - \boldsymbol{q}_1\|} e^{S_2}, \qquad T_4 := -\frac{\boldsymbol{q}_2 - \boldsymbol{q}_1}{\|\boldsymbol{q}_2 - \boldsymbol{q}_1\|} e^{S_2}.$$

Let $\hat{q}(\boldsymbol{z}) = (\mathbf{0}_{8\times 1}; \bar{T}_1; \bar{T}_2)$ where $\bar{T}_i = \frac{T_i}{\sqrt{T_1^2 + T_2^2}}$. Then, $\{e_1, \ldots, e_8, \hat{q}(\boldsymbol{z})\}$ is an orthonormal basis for $\ker M(\boldsymbol{z})$ and $\operatorname{range}((\boldsymbol{J}\hat{q}(\boldsymbol{z}))^\top) \not\subset \ker M(\boldsymbol{z})$. Furthermore, $E \in \mathcal{F}_M$

and $\nabla E(\boldsymbol{z}) = (\boldsymbol{J}\mathcal{P}_M(\boldsymbol{z}))^\top \boldsymbol{c}_M \circ \mathcal{P}_M(\boldsymbol{z})$, where $\mathcal{P}_M(\boldsymbol{z}) = (\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{p}_1, \boldsymbol{p}_2, F_M(\boldsymbol{z}))^\top$, $\boldsymbol{c}_M(\xi) = (\boldsymbol{0}_{4\times 1}; \xi_5; \ldots; \xi_8; 1)$ and $(\boldsymbol{J}\mathcal{P}_M(\boldsymbol{z}))^\top = [e_1, \ldots, e_8, \nabla F_M(\boldsymbol{z})]$.

Proof. The proof directly follows from a straight forward calculation. □

We generate 100 trajectories from $t_0 = 0$ to $t_T = 40$ with $\Delta t = 0.1$ ($T = 400$), whose initial conditions are sampled uniformly from $[0.9, 1.1] \times [-0.1, 0.1] \times [2.1, 2.3] \times [-0.1, 0.1] \times [-0.1, 0.1] \times [1.9, 2.1] \times [0.9, 1.1] \times [-0.1, 0.1] \times [0.9, 1.1] \times [0.1, 0.3]$. We use 80 of them for training and the remaining for testing.

In the top row of Fig 4.6, we again plot one of 20 test trajectories, together with the predicted trajectories of GFINNs in the same way as in the previous example. Since the state variable lies in $\mathbb{R}^{10}$, we plot the predicted trajectories of λ_1, λ_2 and $S_1 + S_2$. We see that the predicted length and entropy of the springs by GFINNs match the ground truth in case 1 and case 2a. In case 2b, due to the chaotic nature of the problem, the predicted trajectory starts to deviate from the ground after certain time.

In the bottom row of Fig 4.6, we report the mean MSE of GFINNs, SPNNs and GNODEs from ten simulations. The shaded areas indicate the maximum and minimum of MSEs as in the gas container example. We clearly observe that in all cases GFINNs can produce better predictive performance compared to other baseline methods. In case 1, all the 10 MSEs of GFINNs are significantly smaller than those of SPNNs. In case 2a and case 2b, the mean MSE of GFINNs is approximately one order of magnitude smaller than GNODEs most of the time. The only exception is in the time window $t \in [30, 40]$, when the MSEs for GFINNs and GNODEs become similar.

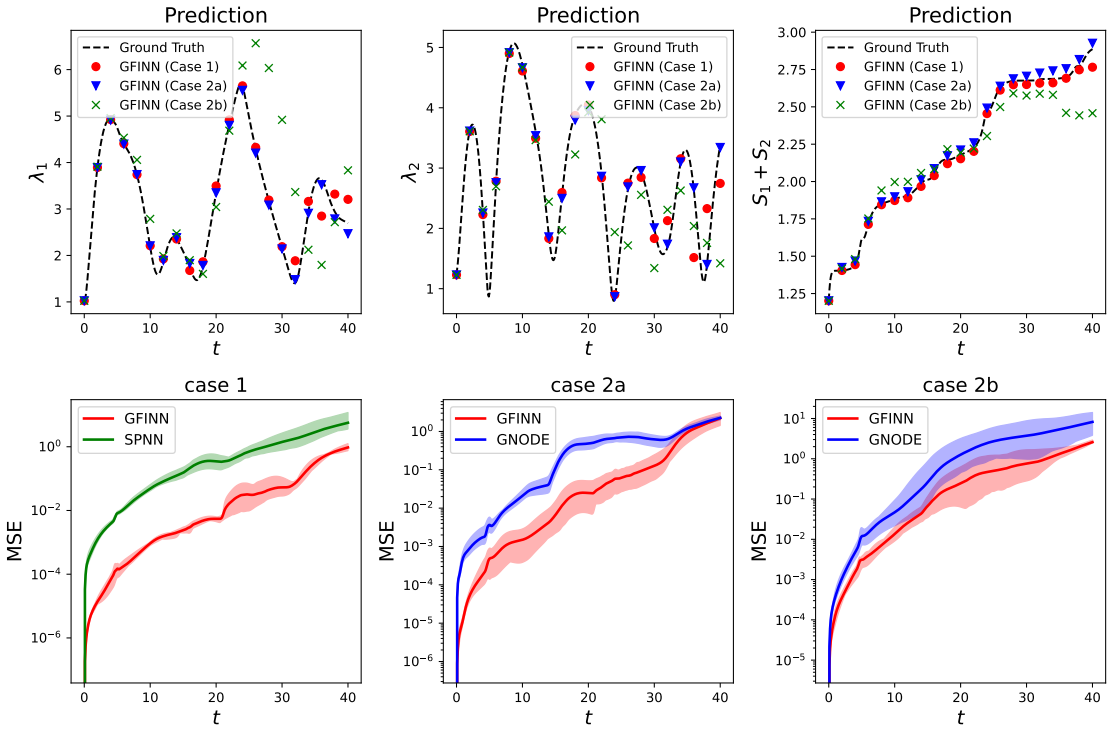


Figure 4.6: **Results for the thermoelastic double pendulum example.** (**Top**) Predicted length of springs λ_1, λ_2 and the predicted total entropy $S_1 + S_2$. Better predictive performance is achieved when there is additional physical knowledge (case 1 and case 2a) beyond the GENERIC formalism. (**Bottom**) The MSE of GFINNs, SPNNs and GNODEs. By incorporating hard constraints into the neural network, GFINNs can produce better predictive performances compared to SPNNs and GNODEs.

4.4.3 Langevin equation

We consider the diffusion of a particle described by the Langevin equation. The state variable vector is $z = (q, p, S_e) \in \mathbb{R}^3$, where q, p are the position and momentum of the particle, respectively, and S_e is the entropy of the surrounding environment. A simple form of energy and entropy is assumed: $E = \frac{p^2}{2} + S_e$ and $S = S_e$. The

Langevin equation is then given by (4.12) with

$$L = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad M(\boldsymbol{z}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{p}{2} \\ 0 & -\frac{p}{2} & \frac{p^2}{2} \end{pmatrix}, \quad \sigma(\boldsymbol{z}) = \begin{pmatrix} 0 \\ 1 \\ -p \end{pmatrix}. \tag{4.19}$$

Note that we choose units such that $k_B = 1$ for notational simplicity.

We simulate 40 trajectories between $t_0 = 0$ to $t_T = 1$ with $\Delta t = 0.004$ for training. The initial state is randomly sampled from a normal distribution with mean $(0, 2, 0)^\top$ and covariance matrix $0.4^2 I$, where I is the identity matrix of size 3. To evaluate the predictive performance of the model, we sample another 50,000 initial conditions from the same distribution, and solve (4.12) using both the ground truth μ, σ and inferred $\mu_{\text{NN}}, \sigma_{\text{NN}}$ with the Euler-Maruyama method. Then we apply the Gaussian kernel density estimator [152] to approximate the distribution of $\boldsymbol{z}$ and $\boldsymbol{z}_{\text{NN}}$ at time steps $t = 0, 0.5, 1, 2$, results of which can be found in Fig. 4.7. Note that we infer dynamics from stochastic samples. In case 1 and case 2a, our model is not only able to recover the correct distribution of $\boldsymbol{z}(t)$, but also gives the correct prediction when extrapolating in time ($t = 2$), using a small amount of data. However, for case 2b and SDENet, even though a similar training error is achieved as shown in Table 4.1, the predicted trajectories do not match the ground truth, which indicates that prior physical knowledge is crucial to make the data-efficient inference. Still, the prediction error of GFINNs is smaller than that of SDENets in all cases as shown in Table 4.1.

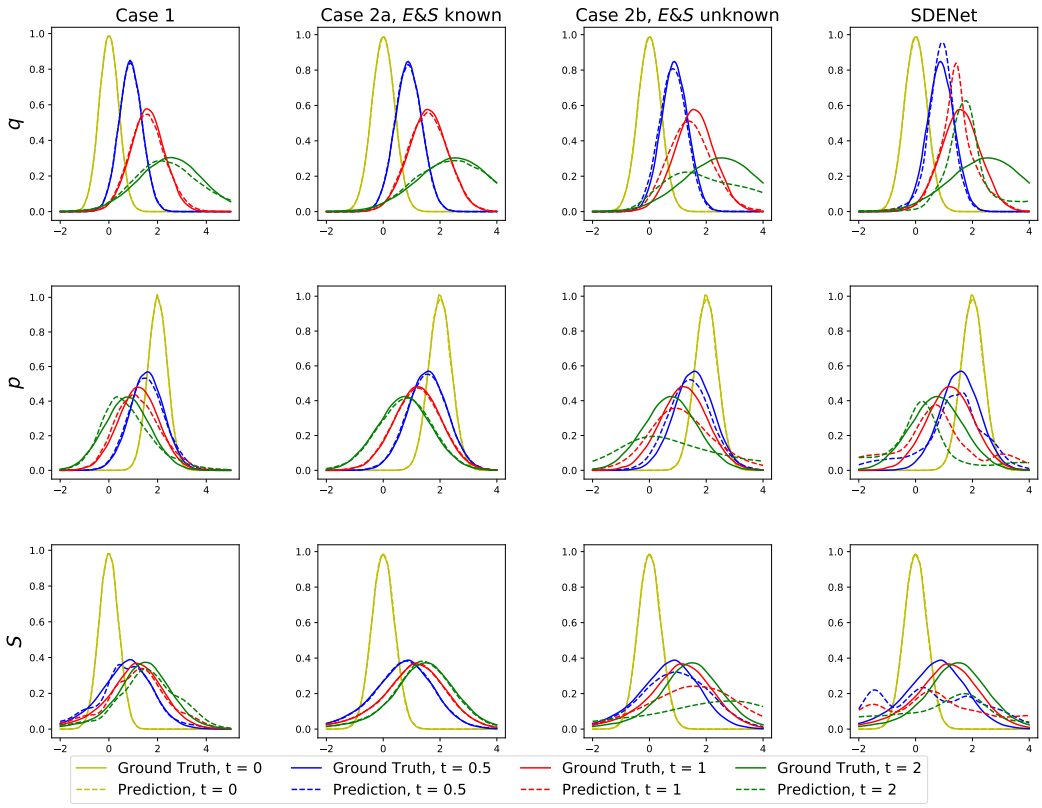


Figure 4.7: **Results for the Langevin equation example.** We plot the predicted and true probability distribution of $\boldsymbol{z}(t)$ at $t = 0, 0.5, 1, 2$ using both types of GFINNs. In case 2a, GFINNs can recover the correct distribution and give the correct prediction when extrapolating in time ($t = 2$). In case 1, there is a small discrepancy between the predicted and true distribution. For case 2b and SDENet, the prediction deviates from the ground truth after certain time.

4.5 Summary

The main contribution of the chapter is to provide a novel neural network surrogate model, GFINN, which can handle the inference and prediction of both deterministic and stochastic irreversible thermodynamic processes using structured neural networks, while strictly preserving the thermodynamics constraints described by the GENERIC formalism at the same time. Theoretically, the proposed architecture, GFINNs, are proven to possess the universality within the corresponding function classes, indicating that the models are general enough to deal with arbitrary data

	Case 1	Case 2a	Case 2b	SDENet
Training loss	-2.7856	-2.7844	-2.6910	-2.7690
Prediction error, $t = 0.5$	1.3×10^{-2}	4.6×10^{-4}	7.6×10^{-2}	4.0×10^{-1}
Prediction error, $t = 1$	6.7×10^{-2}	7.4×10^{-4}	7.4×10^{-1}	1.1

Table 4.1: **Training loss (negative log-likelihood) and prediction error (squared sliced Wasserstein-2 distance) of GFINNs.** Even though the training losses of the three cases are similar, the prediction error shows significant differences, which reflects the different generalization capabilities of different models. In all cases, GFINNs produce a lower prediction error compared to SDENets.

that are solutions to a GENERIC system. Experimentally, we present three examples to support our theoretical findings and successfully demonstrate its advantages over existing methods.

Despite the good performance of GFINNs on low-dimensional systems, it is not experimentally tested whether the proposed method could work well in high dimensions directly, due to the large number of training parameters potentially required. In practice, several Lagrangian particle methods including the smoothed dissipative particle dynamics (SDPD) [59] method can be formulated into the GENERIC framework, normally with high-dimensional state variables. One possible future research direction would be to develop a model that extends the current GFINN framework and scales well in high dimensions. One option would be to use sparse parameterization techniques, as discussed in Section 4.3. Another option would be to combine GFINNs with dimension-reduction techniques like autoencoders, adapting similar ideas from [192, 86, 101, 164].

Appendix

Implementation of architecture

Problem		Gas container		Double pendulum		Langevin equation		
		GFINN	GNODE	GFINN	GNODE	GFINN	SDENet	
Layers	L	-/5/5	-	-/5/5	-	-/5/5	μ	5
	M	-/5/5	-	-/5/5	-	-/5/1		
	E	5/-/5	5	5/-/5	5	5/-/5	σ	5
	S	5/-/1	1	5/-/5	5	5/-/1		
Width	L	-/30/30	-	-/30/30	-	-/30/30	μ	30
	M	-/30/30	-	-/30/30	-	-/30/-		
	E	30/-/30	30	30/-/30	30	30/-/30	σ	30
	S	30/-/-	-	30/-/30	30	30/-/-		

Table 4.2: **Model architecture.** The number of layers and width of the neural network is slightly different across three cases considered in the chapter (case 1, case 2a and case 2b). We list all of them in the table separated by the slash. The layers and width for each component of SPNNs is set to be the same as GFINNs. The hyperbolic tangent (tanh) activation functions are used for all the models. The optimizer is chosen to be Adam [108] with learning rate 0.001. For the gas container and double pendulum examples, we train the models using mini-batch with batch size 100 for 5×10^5 iterations. For the Langevin equation example, we train the models using full batch for 5×10^4 iterations. The important dimension K of skew-symmetric matrices $Q_{S_{\text{NN}}}(\boldsymbol{z})$ and $Q_{E_{\text{NN}}}(\boldsymbol{z})$ in the parameterization of L_{NN} and M_{NN} are chosen to be 5 and 4 respectively in all examples.

GNODEs

The GNODEs [122] parameterize L, M based on bracket structure and use standard neural networks for modelling E, S. The GNODEs' architecture is designed for case 2b and their architecture has a similarity with GFINNs in that two independent neural networks $E_{\text{NN}}, S_{\text{NN}}$ are used to parameterize E and S. However, a key difference lies in the network architectures for L and M. Specifically, GNODEs model L and

M by L^{GNODE} and M^{GNODE} whose (α, β) components are defined by

$$L^{\text{GNODE}}_{\alpha\beta}(\boldsymbol{z}) = \sum_{\gamma} \xi_{\alpha\beta\gamma} \nabla S_{\text{NN}}(\boldsymbol{z})_{\gamma}, \quad M^{\text{GNODE}}_{\alpha\beta}(\boldsymbol{z}) = \sum_{\mu,\nu,m,n} \Lambda^{m}_{\alpha\beta} D_{mn} \Lambda^{n}_{\mu\nu} \nabla E_{\text{NN}}(\boldsymbol{z})_{\beta} \nabla E_{\text{NN}}(\boldsymbol{z})_{\nu}$$

where ξ is a 3d skew-symmetric tensor, Λ^m are 2d skew-symmetric matrices, D is a positive semi-definite matrix. Due to the bracket structure, the GENERIC conditions are enforced under such parameterization. However, L^{GNODE} and M^{GNODE} are merely functions of $\nabla S_{\text{NN}}(\boldsymbol{z})$ and $\nabla E_{\text{NN}}(\boldsymbol{z})$ respectively, which can be seen as the underlying model assumption that differs from GFINNs. In examples where such assumption holds, GNODEs may achieve good performance because they incorporate stronger physical prior knowledge. However, when A depends on not only $\nabla G(\boldsymbol{z})$ but also on some other functions of $\boldsymbol{z}$, GNODEs cannot represent the underlying governing equations due to the lack of expressivity. In [122], the mean squared error (4.2) is also used as the loss function.

SPNNs

[87] proposed SPNNs, which parameterize the gradient of the energy and entropy assuming L and M are known (case 1). The loss function for SPNNs is defined as

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{traj}}} \sum_{k=1}^{N_{\text{traj}}} \frac{1}{T} \sum_{j=1}^{T} \Bigg(\left\| \boldsymbol{z}_{\text{NN}}(t_j; \boldsymbol{z}_0^{(k)}; \theta) - \boldsymbol{z}(t_j; \boldsymbol{z}_0^{(k)}) \right\|^2 + \lambda \left(\|L(dS)_{\text{NN}}\|^2 + \|M(dE)_{\text{NN}}\|^2 \right) \Bigg),$$

where $(dE)_{\text{NN}}$ and $(dS)_{\text{NN}}$ are standard feed-forward neural networks (FNNs) parameterizing ∇E and ∇S, $L(dS)_{\text{NN}}$ and $M(dE)_{\text{NN}}$ are both evaluated at $\boldsymbol{z}(t_j; \boldsymbol{z}_0^{(k)})$, λ is the hyperparameter which controls the scale of the soft penalty, and $\boldsymbol{z}_{\text{NN}}(t_j; \boldsymbol{z}_0^{(k)}; \theta)$

is computed by applying a numerical integrator to the equation $\dot{\boldsymbol{z}}(t) = F_{\text{NN}}(\boldsymbol{z};\theta) = L(\boldsymbol{z})(dE)_{\text{NN}}(\boldsymbol{z}) + M(\boldsymbol{z})(dS)_{\text{NN}}(\boldsymbol{z})$ starting at $\boldsymbol{z}(t_{j-1};\boldsymbol{z}_0^{(k)})$,

However, the architecture we used for comparison is slightly different from that in the original paper, i.e., we model E, S instead of ∇E and ∇S as neural networks. By parameterizing E, S as neural networks, the surrogate model can learn a dynamical system more effectively, due to the inductive bias that ∇E and ∇S should satisfy the Clairaut's theorem. Another reason for our choice is that we parameterize E, S as neural networks in GFINNs, and the comparison should be made fair by control of variables.

In this chapter we choose λ from $\{0.01, 0.1, 1\}$ and pick the ones that give the lowest MSEs in the two deterministic problems.

SDENet

The SDENets parameterize the drift and diffusion term μ and σ as two independent FNNs and use the negative log-likelihood function (4.14) as the loss function.

Chapter Five

SympOCNet: Solving optimal control problems with applications to high-dimensional multi-agent path planning problems

5.1 Introduction

Optimal control methods are widely applied in many practical problems, which include path planning [34, 170, 88, 46, 151, 120, 135], humanoid robot control [58, 66, 114, 74, 63, 47], and robot manipulator control [123, 96, 107, 127, 25]. With the increasing impact of drones in everyday tasks as well as specialized military tasks, the path planning problem of multiple drones has become increasingly important. However, it is challenging to model this problem using the optimal control formulation because the dimensionality is usually very high, causing difficulty in applying traditional numerical methods, such as the very accurate pseudospectral (PS) method [61, 62, 160, 167, 76], to solve it. For instance, for a multi-agent path planning problem with M drones, the motion of each drone has to be described by a state variable in $\mathbb{R}^m$, therefore the dimension of the corresponding optimal control problem will be Mm, which is huge when the number of drones M is large. Computational complexity of traditional numerical methods may scale exponentially with respect to the dimension, which makes the high-dimensional problems intractable. This is an open issue called "curse-of-dimensionality" (CoD) [12]. Hence, new approaches are required to tackle high-dimensional optimal control problems efficiently.

In literature, there are some algorithms for solving high-dimensional optimal control problems (or the corresponding Hamilton-Jacobi PDEs), which include optimization methods [38, 44, 41, 43, 28, 27, 191, 121, 110], max-plus methods [2, 3, 56, 73, 77, 138, 137, 139, 140], tensor decomposition techniques [55, 92, 182], sparse grids [17, 75, 105], polynomial approximation [103, 104], model order reduction [5, 115], optimistic planning [16], dynamic programming and reinforcement learning [26, 24, 4, 15, 197], as well as methods based on neural networks [8, 9, 54, 95, 85, 94, 93, 119, 146, 161, 168, 179, 124, 40, 42, 39, 144, 145, 101, 102, 148].

However, path planning problems are in general hard to solve and cannot be solved directly by applying most of the aforementioned algorithms. One difficulty comes from the state constraints given by the complicated environment. Therefore, solving high-dimensional optimal control problems with complicated state constraints is an important yet challenging open problem. The difficulty of high dimensionality can be avoided by solving the sequential path planning problem instead of the simultaneous planning problem [26, 24, 165]. The main idea of sequential planning is to assign for different drones different levels of priorities and then sequentially solve the low-dimensional path planning problem involving each drone according to its priority. However, sequential planning only provides a feasible solution to the original simultaneous planning problem, which may not be optimal. Hence, how to solve high-dimensional simultaneous planning problems with complicated state constraints is still an unsolved problem.

Recently, neural networks have achieved great success in solving high-dimensional PDEs and inverse problems and bear some promise in overcoming the CoD. We can make further progress if we take advantage of the knowledge about the underlying data generation process and encode such information in the neural network architectures and the loss functions [101, 195, 81, 142]. As a specific example, the physics-informed neural network (PINN), which was first introduced in [158], encodes physical information directly into the loss function of neural networks using the residuals of ODEs/PDEs and automatic differentiation. In this work, we leverage PINNs and propose a novel method using a Symplectic optimal control network called SympOCNet to solve high-dimensional optimal control problems. SympOCNet is designed based on the Symplectic network (SympNet), which is a neural network architecture proposed in [102] and is able to approximate arbitrary symplectic maps. In the original paper, SympNet is used to solve the inverse parameter identification problem by

acting as a surrogate model for the data generation procedure. In this chapter, we present a novel way to utilize SympNet for solving Hamiltonian ODEs in a forward manner. By applying the SympOCNet, we encode our knowledge about optimal control problems and Hamilton-Jacobi theory in the neural network. We then apply SympOCNet on multi-agent path planning problems and show its effectiveness and efficiency in solving high-dimensional problems. Specifically, we can solve a path planning problem involving 256 drones, which corresponds to a 512-dimensional optimal control problem.

The organization of this chapter is given as follows. In Section 5.2, we briefly summarize some preliminary background which will be used in the chapter, including Hamiltonian systems and symplectic maps in Section 5.2.1 and SympNet in Section 5.2.2. In Section 5.3, we present our proposed SympOCNet method for the optimal control problems without state constraints, which serves as a building block for the full version of the SympOCNet method. The full version is then introduced in Section 5.4, which is proposed for solving the optimal control problems with state constraints. To be specific, we propose four loss functions, which are compared in the numerical experiments. Our numerical experiments on multi-agent path planning problems are presented in Section 5.5. The first example in Section 5.5.1 is used to compare the four loss functions. The second example in Section 5.5.2 demonstrates the generalizability of our method when only partial data is available in the training process. Then, in Sections 5.5.3 and 5.5.4, we demonstrate the effectiveness and efficiency of SympOCNet on high-dimensional problems, with dimensions ranging from 64 to 512. A summary is provided in Section 5.6.

5.2 Preliminary background

In this chapter, we solve optimal control problems with SympOCNet, a neural network with SympNet architecture and loss function given by the corresponding Hamiltonian ODEs. This section provides some mathematical materials used in the remainder of the chapter. In Section 5.2.1, we provide a brief summary about symplectic maps, Hamiltonian ODEs and their relations. Then, in Section 5.2.2, we review the SympNet architectures. For more details regarding the theory and applications of symplectic methods, we refer the readers to [84].

5.2.1 Hamiltonian systems and symplectic maps

Definition 22 (Symplectic maps)**.** Let U be an open set in $\mathbb{R}^{2n}$. A differentiable map $\phi : U \to \mathbb{R}^{2n}$ is called symplectic if the Jacobian matrix $\nabla\phi$ satisfies

$$\nabla\phi^\top(\boldsymbol{z})J\nabla\phi(\boldsymbol{z}) = J \quad \forall \boldsymbol{z} \in \mathbb{R}^{2n}, \tag{5.1}$$

where J is a matrix with $2n$ rows and $2n$ columns defined by

$$J := \begin{pmatrix} \mathbf{0} & I_n \\ -I_n & \mathbf{0} \end{pmatrix}, \tag{5.2}$$

and I_n denotes the identity matrix with n rows and n columns.

The Hamiltonian ODE system is a dynamical system taking the form

$$\dot{\boldsymbol{z}}(s) = J\nabla H(\boldsymbol{z}(s)),$$

where J is the matrix defined in (5.2) and $H : U \to \mathbb{R}$ is a function called Hamiltonian. The Hamiltonian systems and symplectic maps are highly related to each other. To be specific, the Hamiltonian structure is preserved under the change of variable using any symplectic map. This result is stated in the following theorem.

Theorem 15. [84, Theorem 2.8 on p. 187] Let U and V be two open sets in $\mathbb{R}^{2n}$. Let $\phi : U \to V$ be a change of coordinates such that ϕ and ϕ^{-1} are continuously differentiable functions. If ϕ is symplectic, the Hamiltonian ODE system $\dot{\boldsymbol{z}}(s) = J\nabla H(\boldsymbol{z}(s))$ can be written in the new variable $\boldsymbol{w} = \phi(\boldsymbol{z})$ as

$$\dot{\boldsymbol{w}}(s) = J\nabla \tilde{H}(\boldsymbol{w}(s)), \tag{5.3}$$

where the new Hamiltonian $\tilde{H}$ is defined by

$$\tilde{H}(\boldsymbol{w}) = H(\boldsymbol{z}) = H(\phi^{-1}(\boldsymbol{w})) \quad \forall \boldsymbol{w} \in V. \tag{5.4}$$

Conversely, if $\phi : U \to V$ is a change of coordinates that transforms every Hamiltonian system to another Hamiltonian system by (5.3) and (5.4), then ϕ is symplectic.

Theorem 15 indicates that a symplectic map can transform a Hamiltonian ODE system to another system which is again Hamiltonian and potentially in a simpler form. This is the starting point of our proposed method. It is well known from literature that the optimal trajectory of an optimal control problem is related to a Hamiltonian ODE system under certain assumptions [60]. Such systems may be solved more easily if written in the right coordinates. We propose to learn the corresponding coordinate transformation through a parameterized family of symplectic maps ϕ_θ, where θ represents the unknown parameters to be learned, and solve the Hamiltonian ODEs with new coordinates. The solution to the original problem can be obtained by mapping the trajectory back to original phase space through

$\varphi_\theta = \phi_\theta^{-1}$.

5.2.2 Symplectic networks (SympNets)

SympNet is a neural network architecture proposed in [102] to approximate symplectic transformations. There are different kinds of SympNet architectures. In this chapter, we use the G-SympNet as the class of our symplectic transformation φ_θ. For other architectures, we refer the readers to [102].

Define a function $\hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}\colon \mathbb{R}^n \to \mathbb{R}^n$ by $\hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{x}) := K^\top(\boldsymbol{a} \odot \sigma(K\boldsymbol{x} + \boldsymbol{b}))$ for any $\boldsymbol{x} \in \mathbb{R}^n$, where σ is the activation function, and $\odot$ denotes the componentwise multiplication. Here $\boldsymbol{a}, \boldsymbol{b}$ are vectors in $\mathbb{R}^l$, K is a matrix with l rows and n columns. Any G-SympNet is an alternating composition of the following two parameterized functions:

$$\begin{aligned} \mathcal{G}_{up}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} &= \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p} + \hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{x})\end{pmatrix} \quad \forall \boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n, \\ \mathcal{G}_{low}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} &= \begin{pmatrix}\hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{p}) + \boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} \quad \forall \boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n, \end{aligned} \tag{5.5}$$

where the learnable parameters are the matrix K and the vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^l$. The dimension l (which is the dimension of $\boldsymbol{a}, \boldsymbol{b}$ as well as the number of rows in K) is a positive integer that can be tuned, called the width of SympNet. In [102], it is proven that G-SympNets are universal approximators within the family of symplectic maps. Note that it is easy to obtain the inverse map of a G-SympNet, since we have explicit

formulas for $\mathcal{G}_{up}^{-1}$ and $\mathcal{G}_{low}^{-1}$ given as follows

$$\mathcal{G}_{up}^{-1}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix}=\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}-\hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{x})\end{pmatrix},\quad \mathcal{G}_{low}^{-1}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix}=\begin{pmatrix}\boldsymbol{x}-\hat{\sigma}_{K,\boldsymbol{a},\boldsymbol{b}}(\boldsymbol{p})\\ \boldsymbol{p}\end{pmatrix}. \tag{5.6}$$

5.3 SympOCNet for optimal control problems without state constraints

In this section, we consider the following optimal control problem without state constraints

$$\begin{aligned}&\min\left\{\int_0^T \ell(\boldsymbol{x}(s),\boldsymbol{v}(s))ds\colon \boldsymbol{x}(0)=\boldsymbol{x}_0,\boldsymbol{x}(T)=\boldsymbol{x}_T,\dot{\boldsymbol{x}}(s)=\boldsymbol{v}(s)\in U\,\forall s\in(0,T)\right\}\\ =\ &\min\left\{\int_0^T \ell(\boldsymbol{x}(s),\dot{\boldsymbol{x}}(s))ds\colon \boldsymbol{x}(0)=\boldsymbol{x}_0,\boldsymbol{x}(T)=\boldsymbol{x}_T,\dot{\boldsymbol{x}}(s)\in U\,\forall s\in(0,T)\right\},\end{aligned} \tag{5.7}$$

where $\boldsymbol{v}\colon[0,T]\to U$ is the control function taking values in the control set $U\subseteq\mathbb{R}^n$, $\boldsymbol{x}\colon[0,T]\to\mathbb{R}^n$ is the corresponding trajectory, and $\ell\colon\mathbb{R}^n\times U\to\mathbb{R}$ is called running cost.

It is well-known in the literature (see, for instance, [10]) that the solution $\boldsymbol{x}$ to the optimal control problem (5.7) satisfies the following Hamiltonian ODE system

$$\begin{cases}\dot{\boldsymbol{x}}(s)=\nabla_{\boldsymbol{p}}H(\boldsymbol{x}(s),\boldsymbol{p}(s)), & s\in[0,T],\\ \dot{\boldsymbol{p}}(s)=-\nabla_{\boldsymbol{x}}H(\boldsymbol{x}(s),\boldsymbol{p}(s)), & s\in[0,T],\\ \boldsymbol{x}(0)=\boldsymbol{x}_0,\\ \boldsymbol{x}(T)=\boldsymbol{x}_T,\end{cases} \tag{5.8}$$

where the Hamiltonian $H\colon \mathbb{R}^{2n} \to \mathbb{R}$ is defined by

$$H(\boldsymbol{x}, \boldsymbol{p}) = \sup_{\boldsymbol{v} \in U} \{\langle \boldsymbol{v}, \boldsymbol{p} \rangle - \ell(\boldsymbol{x}, \boldsymbol{v})\}, \quad \forall \boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n. \tag{5.9}$$

A method to solve the optimal control problem (5.7) in high dimensions is to solve the corresponding Hamiltonian ODE system (5.8) instead. However, when the Hamiltonian is too complicated, the Hamiltonian ODEs may be hard to solve. To tackle this difficulty, we propose a method called SympOCNet, which uses the SympNet architecture to solve the optimal control problem (5.7) in high dimensions. The key idea is to perform a change of variables in the phase space using a symplectic map represented by a SympNet, and then solve the Hamiltonian ODEs in the new coordinate system.

Given a symplectic map $\phi\colon \mathbb{R}^{2n} \to \mathbb{R}^{2n}$, we define the change of variables $(\boldsymbol{y}, \boldsymbol{q}) = \phi(\boldsymbol{x}, \boldsymbol{p})$ for any $\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n$. If the phase trajectory $t \mapsto (\boldsymbol{x}(t), \boldsymbol{p}(t))$ is the solution to the Hamiltonian ODEs (5.8), then by Theorem 15, the phase trajectory $t \mapsto (\boldsymbol{y}(t), \boldsymbol{q}(t))$ under the new coordinate system solves the Hamiltonian ODEs with a new Hamiltonian $\tilde{H}\colon \mathbb{R}^{2n} \to \mathbb{R}$ defined by

$$\tilde{H}(\boldsymbol{y}, \boldsymbol{q}) = H(\phi^{-1}(\boldsymbol{y}, \boldsymbol{q})) \quad \forall \boldsymbol{y}, \boldsymbol{q} \in \mathbb{R}^n.$$

In other words, the function $t \mapsto (\boldsymbol{y}(t), \boldsymbol{q}(t))$ satisfies

$$\begin{cases} \dot{\boldsymbol{y}}(s) = \nabla_{\boldsymbol{q}} \tilde{H}(\boldsymbol{y}(s), \boldsymbol{q}(s)), & s \in [0, T], \\ \dot{\boldsymbol{q}}(s) = -\nabla_{\boldsymbol{y}} \tilde{H}(\boldsymbol{y}(s), \boldsymbol{q}(s)), & s \in [0, T], \\ (\phi^{-1})^{(1)}(\boldsymbol{y}(0), \boldsymbol{q}(0)) = \boldsymbol{x}_0, \\ (\phi^{-1})^{(1)}(\boldsymbol{y}(T), \boldsymbol{q}(T)) = \boldsymbol{x}_T, \end{cases} \tag{5.10}$$

where $(\phi^{-1})^{(1)}$ denotes the first n components in ϕ^{-1}. Here, we assume that the new Hamiltonian $\tilde{H}$ has a simpler form such that the corresponding Hamiltonian ODE system (5.10) is easier to solve. To be specific, we assume that $\tilde{H}$ does not depend on the variable $\boldsymbol{y}$, and hence the Hamiltonian ODEs become

$$\begin{cases} \dot{\boldsymbol{y}}(s) = \nabla \tilde{H}(\boldsymbol{q}(s)), & s \in [0, T], \\ \dot{\boldsymbol{q}}(s) = 0, & s \in [0, T], \\ (\phi^{-1})^{(1)}(\boldsymbol{y}(0), \boldsymbol{q}(0)) = \boldsymbol{x}_0, \\ (\phi^{-1})^{(1)}(\boldsymbol{y}(T), \boldsymbol{q}(T)) = \boldsymbol{x}_T. \end{cases} \tag{5.11}$$

The solution $s \mapsto \boldsymbol{q}(s)$ is a constant function, and the solution $s \mapsto \boldsymbol{y}(s)$ is an affine function. Therefore, the solution $s \mapsto (\boldsymbol{y}(s), \boldsymbol{q}(s))$ can be represented using three parameters in $\mathbb{R}^n$, which are denoted by $\boldsymbol{q}_0 = \boldsymbol{q}(0)$, $\boldsymbol{y}_0 = \boldsymbol{y}(0)$ and $\boldsymbol{u} = \dot{\boldsymbol{y}}(s)$. With these three parameters, the solution to (5.11) is given by $\boldsymbol{y}(s) = \boldsymbol{y}_0 + s\boldsymbol{u}$ and $\boldsymbol{q}(s) = \boldsymbol{q}_0$ for any $s \in [0, T]$.

Under this framework, we explain our proposed method in the remainder of this section. In Section 5.3.1, we apply the SympOCNet method to approximate the unknown function ϕ and the unknown parameters $\boldsymbol{u}, \boldsymbol{y}_0$ and $\boldsymbol{q}_0$. Then, after the training process, to further enhance the accuracy we can also apply the pseudospectral method for post-processing (for low-dimensional problems), as we explain in Section 5.3.2.

5.3.1 SympOCNet method

In this section, we describe how to apply SympOCNet method to solve the optimal control problem (5.7). SympOCNet uses a SympNet architecture to represent the

inverse of the unknown function ϕ, which is also a symplectic map. In other words, we approximate ϕ^{-1} using a SympNet denoted by $\varphi_\theta = (\varphi_\theta^{(1)}, \varphi_\theta^{(2)})$, where θ are the trainable parameters inside the neural networks. Here, $\varphi_\theta^{(1)}$ denotes the first n components in φ_θ, which is a map from $(\boldsymbol{y}, \boldsymbol{q})$ to $\boldsymbol{x}$, and $\varphi_\theta^{(2)}$ denotes the last n components in φ_θ, which is a map from $(\boldsymbol{y}, \boldsymbol{q})$ to $\boldsymbol{p}$. Other learnable parameters include $\boldsymbol{u}, \boldsymbol{y}_0, \boldsymbol{q}_0 \in \mathbb{R}^n$ mentioned above, which respectively correspond to $\dot{\boldsymbol{y}}(s)$, $\boldsymbol{y}(0)$ and $\boldsymbol{q}(s)$ in the new Hamiltonian ODE system (5.11). With the SympNet φ_θ and the variables $\boldsymbol{u}, \boldsymbol{y}_0, \boldsymbol{q}_0$, the solution to the original Hamiltonian ODEs (5.8) is given by $s \mapsto \varphi_\theta(\boldsymbol{y}_0 + s\boldsymbol{u}, \boldsymbol{q}_0)$.

To learn the SympNet φ and the parameters $\boldsymbol{u}, \boldsymbol{y}_0$ and $\boldsymbol{q}_0$, we apply the PINN loss [158] on the Hamiltonian ODEs (5.8), and get

$$\begin{aligned}
\mathcal{L}_{res} &= \sum_{j=1}^{N} \|\frac{d\boldsymbol{x}(s_j)}{ds} - \nabla_{\boldsymbol{p}} H(\boldsymbol{x}(s_j), \boldsymbol{p}(s_j))\|^2 + \sum_{j=1}^{N} \|\frac{d\boldsymbol{p}(s_j)}{ds} + \nabla_{\boldsymbol{x}} H(\boldsymbol{x}(s_j), \boldsymbol{p}(s_j))\|^2 \\
&= \sum_{j=1}^{N} \|\nabla_{\boldsymbol{y}} \varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\boldsymbol{u} - \nabla_{\boldsymbol{p}} H(\varphi_\theta(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0))\|^2 \\
&\quad + \sum_{j=1}^{N} \|\nabla_{\boldsymbol{y}} \varphi_\theta^{(2)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\boldsymbol{u} + \nabla_{\boldsymbol{x}} H(\varphi_\theta(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0))\|^2,
\end{aligned}$$

where the second equality holds by the following chain rule

$$\begin{aligned}
\frac{d\boldsymbol{x}(s_j)}{ds} &= \nabla_{\boldsymbol{y}} \boldsymbol{x}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\frac{d\boldsymbol{y}(s_j)}{ds} + \nabla_{\boldsymbol{q}} \boldsymbol{x}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\frac{d\boldsymbol{q}(s_j)}{ds} \\
&= \nabla_{\boldsymbol{y}} \varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\boldsymbol{u}, \\
\frac{d\boldsymbol{p}(s_j)}{ds} &= \nabla_{\boldsymbol{y}} \boldsymbol{p}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\frac{d\boldsymbol{y}(s_j)}{ds} + \nabla_{\boldsymbol{q}} \boldsymbol{p}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\frac{d\boldsymbol{q}(s_j)}{ds} \\
&= \nabla_{\boldsymbol{y}} \varphi_\theta^{(2)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)\boldsymbol{u}.
\end{aligned}$$

Here, the data points are given by $s_1, \ldots, s_N \in [0, T]$. The gradients $\nabla_{\boldsymbol{y}} \varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)$ and $\nabla_{\boldsymbol{y}} \varphi_\theta^{(2)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0)$ are calculated by the automatic differentiation of

the SympNet φ. Moreover, we have another loss term for the initial and terminal conditions of $\boldsymbol{x}$, defined by

$$\begin{aligned}\mathcal{L}_{bd} &= \|\boldsymbol{x}(0) - \boldsymbol{x}_0\|^2 + \|\boldsymbol{x}(T) - \boldsymbol{x}_T\|^2 \\ &= \|\varphi_\theta^{(1)}(\boldsymbol{y}_0, \boldsymbol{q}_0) - \boldsymbol{x}_0\|^2 + \|\varphi_\theta^{(1)}(\boldsymbol{y}_0 + T\boldsymbol{u}, \boldsymbol{q}_0) - \boldsymbol{x}_T\|^2.\end{aligned}$$

The total loss function $\mathcal{L}_{ODE}$ is a weighted sum of the two loss terms, defined by

$$\mathcal{L}_{ODE} = \mathcal{L}_{res} + \lambda \mathcal{L}_{bd}, \tag{5.12}$$

where λ is a positive hyperparameter. Based on our empirical observation, the result given by SympOCNet is not very sensitive to the choice of λ, especially when we also enforce the initial and terminal conditions through the augmented Lagrangian method described in Section 5.4.3. After training, the optimal trajectory is obtained from

$$\boldsymbol{x}(s) = \varphi_\theta^{(1)}(\boldsymbol{y}_0 + s\boldsymbol{u}, \boldsymbol{q}_0) \quad \forall s \in [0, T], \tag{5.13}$$

where the function $\varphi_\theta^{(1)}$ contains the first n components of the SympNet φ_θ. An illustration of our proposed SympOCNet method is shown in Figure 5.1.

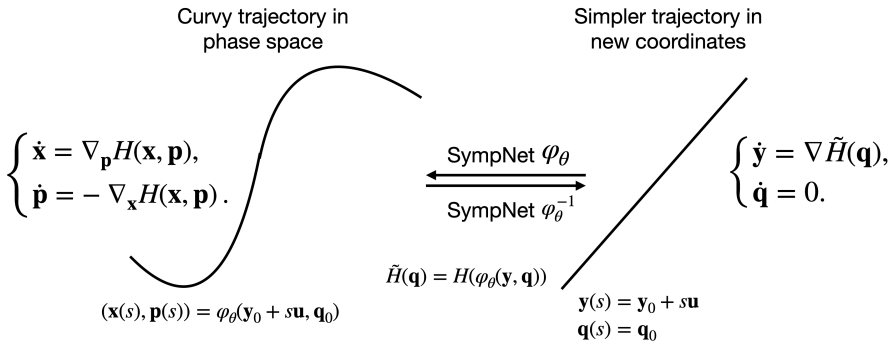


Figure 5.1: **An illustration of the SympOCNet method.** We propose to use a SympNet φ_θ^{-1} to map the curvy trajectory $(\boldsymbol{x}(t), \boldsymbol{p}(t))$ in the phase space to simpler trajectory in new coordinates $(\boldsymbol{y}, \boldsymbol{q})$ and solve the corresponding Hamiltonian system of the optimal control problem.

5.3.2 Post-processing using the pseudospectral method

In Section 5.3.1, we introduced the SympOCNet to solve the optimal control problem (5.7). In this section, we explain how to apply the pseudospectral method to postprocess SympOCNet results to improve feasibility and optimality. The pseudospectral method is a popular method proposed to solve a general optimal control problem [61, 62, 160, 167, 76]. It applies the quadrature rule to discretize the integration and the ODE in a continuous optimal control problem. In this way, the continuous optimal control problem is converted to a finite-dimensional optimization problem with constraints, denoted by (P_N), where N is the number of the collocation points. Some optimization algorithms such as sequential quadratic programming are applied to solve the finite-dimensional problem (P_N).

Note that if the optimization algorithm used for solving the finite-dimensional problem (P_N) provides a feasible solution, then the obtained trajectory is also feasible on every collocation point. Therefore, in the next section when we have state constraints in the optimal control problem, if the optimization algorithm provides a feasible solution to (P_N), then the pseudospectral method gives a trajectory that satisfies the state constraints on the collocation points. Moreover, the convergence of the pseudospectral method as N goes to infinity is proved in the literature (see [167, Section 4.3] and the references therein). To be specific, assuming the state space and the control space are both bounded, as N goes to infinity, the optimal solution of (P_N) has a convergent subsequence whose limit is the optimal solution of the original optimal control problem (5.7).

Although the pseudospectral method has a theoretical guarantee, numerically computing the optimal solution of (P_N) is in general a hard task, especially when the dimension n and the collocation number N are large. The difficulty comes

from the number of variables and constraints in the optimization problem (P_N), as well as the non-convexity of the problem. When the initialization is good, the high-dimensional non-convex optimization problem (P_N) may be easier to solve. Therefore, the pseudospectral method can be applied as a postprocessing procedure by taking the trajectory obtained from Section 5.3.1 as an initialization.

We will apply this procedure in the numerical experiments in Sections 5.5.1 and 5.5.2. From the numerical results, we observe that the pseudospectral method preserves the shape of the trajectories obtained from the training step of the SympOCNet method, and it makes the trajectories smoother while improving the optimality. Therefore, the pseudospectral method performs well as a post-processing method. It is also observed in the experiments that pseudospectral method itself with a linear initialization may not converge to the correct solution, which justifies the choice of SympOCNet as a good initializer. Note that we do not apply this post-process to the experiments in Sections 5.5.3 and 5.5.4 due to the high dimensionality.

5.4 SympOCNet for optimal control problems with state constraints

In this section, we focus on the following optimal control problem with state constraints

$$\begin{aligned}\min\Bigg\{\int_0^T \ell(\boldsymbol{x}(s),\boldsymbol{v}(s))ds\colon h(\boldsymbol{x}(s))\geq 0, \dot{\boldsymbol{x}}(s)=\boldsymbol{v}(s)\in U\forall s\in[0,T],\\ \boldsymbol{x}(0)=\boldsymbol{x}_0,\boldsymbol{x}(T)=\boldsymbol{x}_T\Bigg\}\\ =\min\Bigg\{\int_0^T \ell(\boldsymbol{x}(s),\dot{\boldsymbol{x}}(s))ds\colon h(\boldsymbol{x}(s))\geq 0, \dot{\boldsymbol{x}}(s)\in U\forall s\in[0,T],\\ \boldsymbol{x}(0)=\boldsymbol{x}_0,\boldsymbol{x}(T)=\boldsymbol{x}_T\Bigg\},\end{aligned} \tag{5.14}$$

where $\ell\colon \mathbb{R}^n\times U\to\mathbb{R}$ is the running cost, and $h\colon \mathbb{R}^n\to\mathbb{R}^m$ is a function providing the constraint on the state variable $\boldsymbol{x}$. For instance, to avoid obstacles, h can be defined using the signed distance function to the obstacles.

In order to apply the SympOCNet method in Section 5.3, we need to convert the original problem (5.14) to an unconstrained optimal control problem, which is achieved using the soft penalty method described in Section 5.4.1. The soft penalty method will enforce the constraints in an asymptotic sense, i.e., when the hyperparameters converge to zero under some conditions, which is impractical in experiments. In the training process, when the hyperparameters in the soft penalty are fixed, to improve the feasibility of the output trajectory, we introduce extra terms in the loss function in Sections 5.4.2 and 5.4.3. The selection of these extra loss terms may differ from problem to problem. Later in Section 5.5.1, we compare the performance of different loss functions under our path planning problem setup to choose a suitable

loss function for the other experiments.

5.4.1 Soft penalty method for the optimal control problem

To convert a constrained problem to an unconstrained one, the soft penalty method replaces the hard constraint by a soft penalty term in the objective function. Here, we consider the log penalty function [169, 180], denoted by $\beta_a \colon \mathbb{R} \to \mathbb{R}$ and defined by

$$\beta_a(x) := \begin{cases} -\log(x), & \text{if } x > a, \\ -\log(a) + \frac{1}{2}\left(\left(\frac{x-2a}{a}\right)^2 - 1\right), & \text{if } x \le a, \end{cases} \tag{5.15}$$

where a is a positive hyperparameter. The high-dimensional log penalty function is the summation of the one-dimensional function acting on each component, i.e., for any $m > 1$, the log penalty function $\beta_a \colon \mathbb{R}^m \to \mathbb{R}$ is defined by

$$\beta_a(\boldsymbol{x}) := \sum_{i=1}^{m} \beta_a(x_i) \quad \forall \boldsymbol{x} = (x_1, \dots, x_m) \in \mathbb{R}^m. \tag{5.16}$$

With this log penalty term, the problem (5.14) is converted to

$$\min \left\{ \int_0^T \ell(\boldsymbol{x}(s), \dot{\boldsymbol{x}}(s)) + \epsilon \beta_a(h(\boldsymbol{x}(s)) ds \colon \dot{\boldsymbol{x}}(s) \in U \forall s \in [0,T], \right. \\ \left. \boldsymbol{x}(0) = \boldsymbol{x}_0, \boldsymbol{x}(T) = \boldsymbol{x}_T \right\}, \tag{5.17}$$

where ϵ and a are positive hyperparameters. This problem (5.17) is in the form of (5.7), and hence can be solved using the SympOCNet method in Section 5.3. By straightforward calculation, for a sequence of parameters ϵ_k and a_k such that they

both converge to zero and the following equations hold

$$\lim_{k\to\infty} \epsilon_k \log a_k = 0, \qquad \lim_{k\to\infty} \frac{a_k^2}{\epsilon_k} = 0,$$

the penalty term $\epsilon_k \beta_{a_k}$ converges to the indicator function of $[0, +\infty)$, which takes the value 0 in $[0, +\infty)$ and $+\infty$ in $(-\infty, 0)$. Therefore, the objective function in (5.17) with parameters ϵ_k, a_k converges to the objective function in (5.14) with the hard constraint given by h.

With this penalty term, the Hamiltonian corresponding to the optimal control problem (5.17) equals

$$H_{\epsilon,a}(\boldsymbol{x}, \boldsymbol{p}) = \sup_{\boldsymbol{v}\in U}\{\langle \boldsymbol{v}, \boldsymbol{p}\rangle - \ell(\boldsymbol{x}, \boldsymbol{v}) - \epsilon\beta_a(h(\boldsymbol{x}))\} = H(\boldsymbol{x}, \boldsymbol{p}) - \epsilon\beta_a(h(\boldsymbol{x})), \tag{5.18}$$

where H is the Hamiltonian corresponding to the running cost ℓ defined in (5.17).

Remark 4. Note that another widely-used method to covert a constrained optimization problem to an unconstrained one is the augmented Lagrangian method. In our problem, the constraint is enforced for any time $s \in [0, T]$. To apply the augmented Lagrangian method to convert the state-constrained problem (5.14) to an optimal control problem without state constraints, the Lagrange multiplier needs to be a function of time, denoted by $\boldsymbol{\lambda}\colon [0, T] \to \mathbb{R}^m$. With the Lagrangian multiplier function $\boldsymbol{\lambda}(\cdot)$, the augmented Lagrangian method converts the state-constrained problem (5.14) to

$$\max_{\boldsymbol{\lambda}(\cdot)} \min_{\boldsymbol{v}(\cdot)} \left\{ \int_0^T \ell(\boldsymbol{x}(s), \boldsymbol{v}(s)) + \frac{1}{2\rho} \| \max\{\mathbf{0}, \boldsymbol{\lambda}(s) - \rho h(\boldsymbol{x}(s))\}\|^2 ds : \right.$$
$$\left. \dot{\boldsymbol{x}}(s) = \boldsymbol{v}(s) \in U, \forall s \in [0, T], \boldsymbol{x}(0) = \boldsymbol{x}_0, \boldsymbol{x}(T) = \boldsymbol{x}_T \right\},$$

where ρ is a positive parameter. Note that the new running cost depends on the time variable, and hence the corresponding Hamiltonian also depends on time. This requires a more complicated symplectic method to handle the time-dependent Hamiltonian. Therefore, we do not apply the augmented Lagrangian method for the constrained optimal control problem (5.14) in this chapter. Instead, we add extra terms in the loss function (see Sections 5.4.2 and 5.4.3) to enforce the state constraint.

5.4.2 Auxiliary penalty in the loss function of SympOCNet

To further enforce the state constraint $h(\boldsymbol{x}(s)) \geq 0$, another way is to add the penalty term in the loss function of the training process. Here, we introduce two penalty terms as potential candidates, which include the log penalty β_a defined in (5.16) and the quadratic penalty. The total loss function in this case is defined by

$$\mathcal{L} = \mathcal{L}_{ODE} + \mathcal{L}_{aux}. \tag{5.19}$$

$\mathcal{L}_{ODE}$ is the ODE loss defined in (5.12), $\mathcal{L}_{aux}$ is the auxiliary loss or the additional penalty term. In case of log penalty corresponding to the state constraint $h(\boldsymbol{x}(s)) \geq 0$, it is defined by

$$\begin{aligned}\mathcal{L}_{aux}(\theta, \boldsymbol{y}_0, \boldsymbol{q}_0, \boldsymbol{u}) = \frac{\tilde{\lambda}}{N} \sum_{j=1}^{N} & \epsilon\beta_a(h(\varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j\boldsymbol{u}, \boldsymbol{q}_0))) \\ & + \epsilon\beta_a(\boldsymbol{x}_0 - \varphi_\theta^{(1)}(\boldsymbol{y}_0, \boldsymbol{q}_0)) + \epsilon\beta_a(\varphi_\theta^{(1)}(\boldsymbol{y}_0, \boldsymbol{q}_0) - \boldsymbol{x}_0) \\ & + \epsilon\beta_a(\boldsymbol{x}_T - \varphi_\theta^{(1)}(\boldsymbol{y}_0 + T\boldsymbol{u}, \boldsymbol{q}_0)) + \epsilon\beta_a(\varphi_\theta^{(1)}(\boldsymbol{y}_0 + T\boldsymbol{u}, \boldsymbol{q}_0) - \boldsymbol{x}_T),\end{aligned} \tag{5.20}$$

where β_a is the penalty function defined in (5.16), $\tilde{\lambda}$ is a positive hyperparameter. Similarly, for quadratic penalty it is defined by

$$\mathcal{L}_{aux}(\theta, \boldsymbol{y}_0, \boldsymbol{q}_0, \boldsymbol{u}) = \frac{\tilde{\lambda}}{N} \sum_{j=1}^{N} \| \min\{h(\varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j \boldsymbol{u}, \boldsymbol{q}_0)), 0\}\|^2, \tag{5.21}$$

where we do not add the penalty term for the initial and terminal conditions, since the quadratic penalty for them are the same with $\mathcal{L}_{bd}$ in the loss function $\mathcal{L}_{ODE}$. We denote the total loss $\mathcal{L}$ as $\mathcal{L}_{log}$ and $\mathcal{L}_{quad}$ when log and quadratic penalty is applied respectively.

5.4.3 Augmented Lagrangian method in the training process of SympOCNet

Another method to enforce the state constraint $h(\boldsymbol{x}(s)) \geq 0$ is to apply augmented Lagrangian method in the training process. The loss function for the augmented Lagrangian method takes the same form as (5.19), with auxiliary loss defined as

$$\begin{aligned}
&\mathcal{L}_{aux}(\theta, \boldsymbol{y}_0, \boldsymbol{q}_0, \boldsymbol{u}, \boldsymbol{\mu}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) \\
=& \frac{1}{2\rho_1} \sum_{j=1}^{N} \| \max\{\boldsymbol{0}, \boldsymbol{\mu}(s_j) - \rho_1 h(\varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j \boldsymbol{u}, \boldsymbol{q}_0))\}\|^2 \\
&+ \frac{1}{2\rho_2} \left(\|\boldsymbol{\lambda}_1 - \rho_2(\boldsymbol{x}_0 - \varphi_\theta^{(1)}(\boldsymbol{y}_0, \boldsymbol{q}_0))\|^2 + \|\boldsymbol{\lambda}_2 - \rho_2(\boldsymbol{x}_T - \varphi_\theta^{(1)}(\boldsymbol{y}_0 + T\boldsymbol{u}, \boldsymbol{q}_0))\|^2 \right),
\end{aligned} \tag{5.22}$$

where $\boldsymbol{\mu}(s_j) \in \mathbb{R}^m$, $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \mathbb{R}^n$ are the augmented Lagrange multipliers for the constraints $h(\boldsymbol{x}(s_j)) \geq 0$, $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and $\boldsymbol{x}(T) = \boldsymbol{x}_T$, respectively, and $\rho_1, \rho_2 > 0$ are positive hyperparameters in the augmented Lagrangian method. We refer to the total loss in this case as $\mathcal{L}_{aug}$. The training process becomes the iterative scheme

in Algorithm 1. In the literature, the convergence of the augmented Lagrangian method is guaranteed under certain assumptions (see, [14, Section 3.1], for instance). However, in our case, it is unclear whether these assumptions hold.

Algorithm 1 Training process using augmented Lagrangian method.

1: **for** $k = 1, \dots, N_{iters}$ **do**
2: Train the SympNet φ_θ and parameters $\boldsymbol{y}_0, \boldsymbol{q}_0, \boldsymbol{u} \in \mathbb{R}^n$ simultaneously for several iterations using the loss function $\mathcal{L}_{aug}(\theta, \boldsymbol{y}_0, \boldsymbol{q}_0, \boldsymbol{u}, \boldsymbol{\mu}^k, \boldsymbol{\lambda}_1^k, \boldsymbol{\lambda}_2^k)$;
3: Update the Lagrange multiplier by

$$\begin{aligned} \boldsymbol{\mu}^{k+1}(s_j) &= \max\{\boldsymbol{0}, \boldsymbol{\mu}^k(s_j) - \rho_1 h(\varphi_\theta^{(1)}(\boldsymbol{y}_0 + s_j \boldsymbol{u}, \boldsymbol{q}_0))\}, \forall j = 1, \dots, N, \\ \boldsymbol{\lambda}_1^{k+1} &= \boldsymbol{\lambda}_1^k - \rho_2(\boldsymbol{x}_0 - \varphi_\theta^{(1)}(\boldsymbol{y}_0, \boldsymbol{q}_0)), \\ \boldsymbol{\lambda}_2^{k+1} &= \boldsymbol{\lambda}_2^k - \rho_2(\boldsymbol{x}_T - \varphi_\theta^{(1)}(\boldsymbol{y}_0 + T\boldsymbol{u}, \boldsymbol{q}_0)). \end{aligned} \tag{5.23}$$

4: Adjust the parameters ρ_1 and ρ_2 in $\mathcal{L}_{aug}$. In our implementation, we apply Algorithm 2 for tuning the parameters.
5: **end for**
6: **return** The trajectory $\boldsymbol{x}(\cdot)$ computed using (5.13).

In practice, we need to tune the hyperparameters ρ_1 and ρ_2 in the last step of each iteration in Algorithm 1. In our implementation, these hyperparameters are tuned automatically using a similar strategy as in [33]. The details are given in Algorithm 2, which is executed once after (5.23) in each iteration of the training process. In Algorithm 2, we set $\eta^* = 0.001$, $\alpha = \beta = 0.5$, $\tau = 1.1$, $\eta_0 = 0.1$.

Note that the augmented Lagrangian method introduced here is different from the one in Remark 4. Remark 4 applies the augmented Lagrangian method to the optimal control problem (5.14), while this section applies the augmented Lagrangian method to the training process (and hence the additional term (5.22) is added to the loss function instead of the running cost).

Algorithm 2 Hyperparameter tuning algorithm which is executed once after (5.23) in each iteration of Algorithm 1.

Input: The Lagrange multipliers $\boldsymbol{\mu}^{k+1}(s_j)$, $\boldsymbol{\lambda}_1^{k+1}$, $\boldsymbol{\lambda}_2^{k+1}$ computed in (5.23), the Lagrange multipliers $\boldsymbol{\mu}^{k}(s_j)$, $\boldsymbol{\lambda}_1^{k}$, $\boldsymbol{\lambda}_2^{k}$ (from the previous iteration in the training process), and other parameters η_1, ρ_1, η_2, ρ_2.
Output: The updated Lagrange multipliers $\boldsymbol{\mu}^{k+1}(s_j)$, $\boldsymbol{\lambda}_1^{k+1}$, $\boldsymbol{\lambda}_2^{k+1}$ and the updated parameters η_1, ρ_1, η_2, ρ_2.
Set parameters $\alpha, \beta, \eta^*, \tau, \eta_0$.
if $\max_{j=1,\dots,N} \|\boldsymbol{\mu}^{k+1}(s_j) - \boldsymbol{\mu}^{k}(s_j)\|_\infty < \rho_1 \max\{\eta^*, \eta_1\}$ **then**
    $\eta_1 \leftarrow \frac{\eta_1}{1+\rho_1^\beta}$;
else
    $\eta_1 \leftarrow \frac{\eta_0}{1+\rho_1^\alpha}$;
    $\rho_1 \leftarrow \tau\rho_1$;
    $\boldsymbol{\mu}^{k+1}(s_j) \leftarrow \boldsymbol{\mu}^{k}(s_j), \forall j = 1, \dots, N$;
end if
if $\max\{\|\boldsymbol{\lambda}_1^{k+1} - \boldsymbol{\lambda}_1^{k}\|_\infty, \|\boldsymbol{\lambda}_2^{k+1} - \boldsymbol{\lambda}_2^{k}\|_\infty\} < \rho_2 \max\{\eta^*, \eta_2\}$ **then**
    $\eta_2 \leftarrow \frac{\eta_2}{1+\rho_2^\beta}$;
else
    $\eta_2 \leftarrow \frac{\eta_0}{1+\rho_2^\alpha}$;
    $\rho_2 \leftarrow \tau\rho_2$;
    $\boldsymbol{\lambda}_1^{k+1} \leftarrow \boldsymbol{\lambda}_1^{k}$; $\boldsymbol{\lambda}_2^{k+1} \leftarrow \boldsymbol{\lambda}_2^{k}$;
end if

5.5 Applications in path planning problems with obstacle and collision avoidance

In this section, we apply our method to path planning problems with multiple drones. We assume that each drone is represented by a ball in the physical space $\mathbb{R}^m$ with radius C_d. Note that the meaning of the notation m in this section (i.e., the dimension of the physical space) is different from m in Section 5.4 (which is the number of state constraints). We set the state variable to be $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) \in \mathbb{R}^{Mm}$, where M is the number of drones, and each $\boldsymbol{x}_j \in \mathbb{R}^m$ denotes the position of the center of each drone. In other words, the state variable is the concatenation of the position variables of all drones. Similarly, the control variable $\boldsymbol{v}$ is the concatenation of the velocity variables for all drones, i.e., $\boldsymbol{v}$ equals $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_M) \in U^M$, where each $\boldsymbol{v}_j$ is the velocity of the j-th drone, and $U \subseteq \mathbb{R}^m$ is the space for all admissible velocities for each drone. In our numerical experiments, the control space U is the ball in $\mathbb{R}^m$ with zero center and radius $C_{\boldsymbol{v}} > 0$, i.e., the norm of the velocity of each drone has the upper bound $C_{\boldsymbol{v}}$.

We want to solve for the optimal trajectory with minimal energy, and hence the running cost is set to be

$$\ell(\boldsymbol{x}, \boldsymbol{v}) = \frac{1}{2}\|\boldsymbol{v}\|^2 \quad \forall \boldsymbol{x} \in \mathbb{R}^{Mm}, \boldsymbol{v} \in U^M.$$

The corresponding Hamiltonian equals

$$\begin{aligned} H(\boldsymbol{x},\boldsymbol{p}) &= \sup_{\boldsymbol{v}\in U^M}\left\{\langle \boldsymbol{v},\boldsymbol{p}\rangle - \frac{1}{2}\|\boldsymbol{v}\|^2\right\} = \sum_{i=1}^{M}\sup_{\boldsymbol{v}_i\in U}\left\{\langle \boldsymbol{v}_i,\boldsymbol{p}_i\rangle - \frac{1}{2}\|\boldsymbol{v}_i\|^2\right\} \\ &= \sum_{i=1}^{M}\begin{cases}\dfrac{1}{2}\|\boldsymbol{p}_i\|^2 & \text{if } \|\boldsymbol{p}_i\|\le C_{\boldsymbol{v}},\\ C_{\boldsymbol{v}}\|\boldsymbol{p}_i\| - \dfrac{C_{\boldsymbol{v}}^2}{2} & \text{if } \|\boldsymbol{p}_i\| > C_{\boldsymbol{v}},\end{cases} \end{aligned} \tag{5.24}$$

for any $\boldsymbol{p} = (\boldsymbol{p}_1, \cdots, \boldsymbol{p}_M) \in \mathbb{R}^{Mm}$, where each $\boldsymbol{p}_i$ is the momentum variable in $\mathbb{R}^m$ for the i-th drone.

To avoid obstacles and collisions among drones, we set the constraint function h to be $h = (h_1, h_2)$, where h_1 is for obstacle avoidance, and h_2 is for avoiding collisions among drones. If there are n_o obstacles, denoted by $E_1, \ldots, E_{n_o}$, then the function $h_1 \colon \mathbb{R}^{Mm} \to \mathbb{R}^{Mn_o}$ is defined by

$$h_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) = (d(\boldsymbol{x}_1), d(\boldsymbol{x}_2), \ldots, d(\boldsymbol{x}_M)) \quad \forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_M \in \mathbb{R}^m, \tag{5.25}$$

where the function $d \colon \mathbb{R}^m \to \mathbb{R}^{n_o}$ is defined by

$$d(\boldsymbol{x}) = (d_1(\boldsymbol{x}), \ldots, d_{n_o}(\boldsymbol{x})) \quad \forall \boldsymbol{x} \in \mathbb{R}^m, \tag{5.26}$$

and each function d_j is defined such that $d_j(\boldsymbol{x}) < 0$ implies the collision of the drone whose center is at the position $\boldsymbol{x}$ with the j-th obstacle E_j. For instance, d_j can be defined using the signed distance function to E_j. The definition of the function d_j depends on the shape of the constraint set E_j, and hence we do not specify it now but postpone its definition to each example in later sections.

The function $h_2 \colon \mathbb{R}^{Mm} \to \mathbb{R}^{M(M-1)/2}$ is the constraint function for collision avoidance; each component of the constraint function gives a constraint for avoiding col-

lision between a pair of drones. Since each drone can be seen as a ball centered at a point in $\mathbb{R}^m$, two drones collide if and only if the distance between their centers is less than the sum of their radii. Hence, we set the k-th component of h_2 to be

$$(h_2)_k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 - (2C_d)^2 \quad \forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_M \in \mathbb{R}^m, \tag{5.27}$$

where C_d is the radius of a drone, and $k = i+(j-1)(j-2)/2$ for any $1 \leq i < j \leq M$ is the corresponding constraint index for the pair (i, j). Note that the constraints may be duplicated to simplify the implementation. As a result, $(h_2)_k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) < 0$ is equivalent to the collision between the i-th and j-th drones. In other words, such defined constraint function h_2 can avoid collisions among drones.

In the following sections, we present several numerical results. In Section 5.5.1, we compare the performance of different loss functions $\mathcal{L}_{ODE}$, $\mathcal{L}_{log}$, $\mathcal{L}_{quad}$ and $\mathcal{L}_{aug}$ to choose a suitable loss function for the remaining experiments. In Section 5.5.2, we present an offline-online training strategy for an optimal control problem whose initial conditions are not known in the training process of the SympNet. This experiment shows the generalizability of SympOCNet. Then, in Section 5.5.3, we demonstrate the performance of SympOCNet on a high-dimensional problem reported in [165]. From the results in Section 5.5.3, we observe that SympOCNet can handle the path planning problem whose state space has dimension 512, and hence the method can potentially mitigate the CoD. In Section 5.5.4, we apply SympOCNet to a swarm path planning problem in [148], and demonstrate good performance and efficiency in path planning problems, where the agents move in a three-dimensional space.

In Sections 5.5.1 and 5.5.2 we use a 6-layer SympNet with 60 hidden neurons in each layer and ReLU activation function. In these two sections, we used the penalized Hamiltonian $H_{\epsilon,a}$ defined in (5.18). In Sections 5.5.3 and 5.5.4 we use

a 6-layer SympNet with 200 hidden neurons in each layer and ReLU activation function. We used the unpenalized Hamiltonian H defined in (5.24). The state constraints are enforced through the augmented Lagrangian methods. In all these experiments, the parameter λ in (5.12) is set to be 600, and the parameter $\tilde{\lambda}$ in (5.20) and (5.21) is set to be 200. By default, the speed limit $C_{\boldsymbol{v}}$ is set to be 25, the time horizon T is 1, and the total number N of the time samples $s_1, \ldots, s_N$ in the loss function is set to be 200. We choose the sample points $s_1, \ldots, s_N$ to be the uniform grid points in the interval $[0, T]$. We use Adam optimizer introduced in [108] to minimize the loss function with respect all trainable parameters simultaneously. All the numerical experiments are run using a shared NVIDIA GeForce RTX 3090 GPU and a shared NVIDIA RTX A6000 GPU. In each experiment, we only show the figure for a part of the testing cases. More numerical results and codes are provided in https://github.com/zzhang222/SympOCNet. Video animations of these examples are available online at https://github.com/zzhang222/SympOCNet/tree/main/saved_results_animation.

5.5.1 Comparison among different loss functions

In this section, we consider the path planning problem with two obstacles and four drones in $\mathbb{R}^2$. Each obstacle $E_j \subset \mathbb{R}^2$ is a convex set containing points whose distance to a line segment (denoted by $l_j \subset \mathbb{R}^2$) is less than a constant C_o. The initial positions of the four drones are $(-2, -2)$, $(2, -2)$, $(2, 2)$, $(-2, 2)$, and their terminal positions are $(2, 2), (-2, 2), (-2, -2), (2, -2)$. Therefore, the optimal control problem (5.14) is an 8-dimensional problem, whose initial state $\boldsymbol{x}_0$ is $(-2, -2, 2, -2, 2, 2, -2, 2)$ and terminal state $\boldsymbol{x}_T$ is $(2, 2, -2, 2, -2, -2, 2, -2)$. The obstacles and the initial positions of the four drones are shown in Figure 5.2.

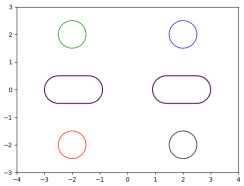
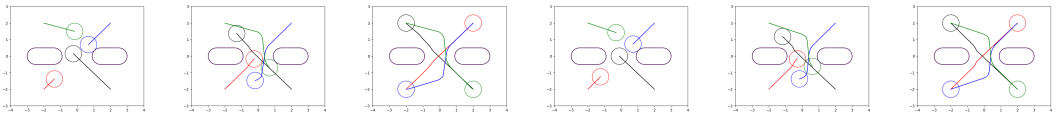


Figure 5.2: **Initial positions of drones and obstacles.** The four drones located at their initial positions are represented by the four colored circles. The two obstacles are represented by round cornered rectangles in the middle.



(a) NN, $t = \frac{1}{3}$ (b) NN, $t = \frac{2}{3}$ (c) NN, $t = 1$ (d) PS, $t = \frac{1}{3}$ (e) PS, $t = \frac{2}{3}$ (f) PS, $t = 1$

Figure 5.3: **Output trajectories with loss $\mathcal{L}_{aug}$ and pseudospectral method.** The left three figures show the trajectories of our proposed SympOCNet method at different time $t = \frac{1}{3}, \frac{2}{3}, 1$, and the right three figures show the outputs of the post-process (pseudospectral method) at different time $t = \frac{1}{3}, \frac{2}{3}, 1$. In each figure, the four colored circles show the positions of the four drones at time t, and the curves connected to the circles show the trajectories before time t.

To avoid the collisions of drones with obstacles, we define the first part of the constraint function h by (5.25), where each component d_j of the function d in (5.26) is defined by

$$d_j(\boldsymbol{x}) = \min_{\boldsymbol{y} \in l_j} \|\boldsymbol{x} - \boldsymbol{y}\|^2 - (C_o + C_d)^2 \quad \forall \boldsymbol{x} \in \mathbb{R}^2.$$

The term $\min_{\boldsymbol{y} \in l_j} \|\boldsymbol{x} - \boldsymbol{y}\|^2$ in d_j gives the squared distance between $\boldsymbol{x}$ and the line segment l_j. Here, we use the squared distance instead of distance function to improve the smoothness of the constraint function h. Since the obstacle E_j contains all points whose distance to the line segment l_j is less than C_o, a drone collide with E_j if and only if the distance between its center and the line segment l_j is less than $C_o + C_d$. Therefore, such defined function d_j provides a constraint which avoids the collisions of drones with the j-th obstacle E_j.

Under this problem setup, we compare four different loss functions $\mathcal{L}_{ODE}$, $\mathcal{L}_{log}$,

$\mathcal{L}_{quad}$, and $\mathcal{L}_{aug}$. With each loss function, we train the neural network for $100,000$ iterations, and then run the pseudospectral method to improve the results. By comparing the performance of the four loss functions in this example, we choose a better loss function for the other experiments in the remainder of this chapter.

The outputs of the four loss functions are shown in Table 5.1, where the hyper-parameters in the penalty term $\epsilon\beta_a(h(\boldsymbol{x}(s))$ are set to be $a = 0.004$ and $\epsilon = 0.0004$. Results for different loss functions are shown in different columns. For each loss function, we repeat the experiment 10 times with different random seeds, and then take average to obtain the statistics in the table. We observe the convergence of the pseudospectral method in all repeated experiments, which shows the robustness of our SympOCNet method, providing optimal or suboptimal solutions in this example. The second to fourth lines in Table 5.1 show the minimal constraint value, the cost value in the optimal control problem, and the running time of the training process of our proposed SympOCNet method. The last four lines in Table 5.1 show the minimal constraint value, the cost value in the optimal control problem, the number of iterations, and the running time of the post-process using pseudospectral method.

loss function		$\mathcal{L}_{ODE}$	$\mathcal{L}_{log}$	$\mathcal{L}_{quad}$	$\mathcal{L}_{aug}$
SympOCNet	min constraint	-0.834	-0.00704	-0.0824	-4.84E-04
	cost	79.62	94.15	76.50	93.98
	running time (s)	1229	1776	1361	1708
PS	min constraint	-4.70E-09	-9.87E-12	-2.04E-12	-8.02E-13
	cost	73.47	91.00	76.95	80.91
	# iterations	49	19	17	23
	running time (s)	104	44	38	47

Table 5.1: **The comparison among results of four loss functions with SympOCNet method and pseudospectral post-process.** We show the minimal constraint value $\min_s h(\boldsymbol{x}(s))$, the cost $\int_{s=0}^{T} \frac{||\boldsymbol{v}(s)||^2}{2} ds$, and the running time of the SympOCNet as well as those statistics after the post-process. The loss function $\mathcal{L}_{aug}$ provides the solution with least amount of constraint violations and reasonable cost values. It can be seen that in all cases, the SympOCNet provides a good initialization for the pseudospectral method.

minimal constraint	cost	number of iterations	running time (s)
-1.00E+00	64.00	5	44

Table 5.2: **The result of the pseudospectral method with the linear initialization.** We show the minimal constraint value $\min_s h(\boldsymbol{x}(s))$, the cost $\int_{s=0}^{T} \frac{||\boldsymbol{v}(s)||^2}{2} ds$, the number of iterations, and the running time of the pseudospectral method with the linear initialization (i.e., the initial trajectory is an affine function of the time variable, and the initial velocity is a constant function). The minimal constraint value is -1.00, which shows that the constraint is not satisfied, and the pseudospectral method with the linear initialization does not converge to a feasible solution.

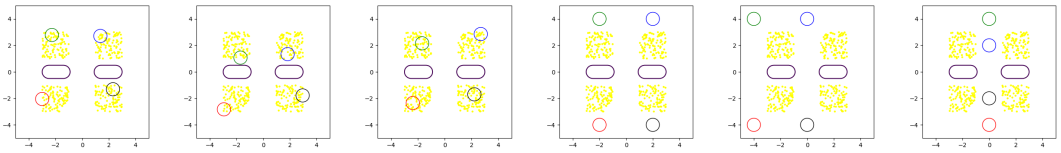
Comparing the results of the four loss functions, we observe that the penalty term and the augmented Lagrangian term in the loss function indeed improve the feasibility of the obtained trajectory. The constraints are better satisfied using the loss $\mathcal{L}_{aug}$ in (5.22), while the corresponding cost value may be a little bit higher than the the results of $\mathcal{L}_{quad}$. In the other experiments, the problems are more complicated, and the constraints are harder to satisfy. Therefore, we use the loss function $\mathcal{L}_{aug}$ for the experiments in the remainder of the chapter to help enforce the state constraints. For simpler problems, other loss function may be more preferable for faster running time. How to choose the loss function adaptively may be a possible future direction. Although the augmented Lagrangian method and log penalty loss function are slightly slower than the other two loss functions, it takes less than 30 minutes to run $100,000$ training iterations, which shows the efficiency of SympOCNet.

To show the improvement using our proposed SympOCNet method as an initialization of the pseudospectral method, we solve the same problem using the pseudospectral method with the linear initialization, i.e., we set the initial guess of the pseudospectral method to be $\boldsymbol{v}(t) = \frac{\boldsymbol{x}_T - \boldsymbol{x}_0}{T}$ and $\boldsymbol{x}(t) = \frac{(\boldsymbol{x}_T - \boldsymbol{x}_0)t}{T}$ for any $t \in [0, T]$. The minimal constraint value, the cost value in the optimal control problem, the number of iterations, and the running time are shown in Table 5.2. The minimal constraint value in Table 5.2 is -1.00, which shows that the pseudospectral method

with the linear initialization does not converge to a feasible solution. Therefore, compared with the linear initialization, our SympOCNet method, no matter which loss function in $\mathcal{L}_{ODE}$, $\mathcal{L}_{log}$, $\mathcal{L}_{quad}$, and $\mathcal{L}_{aug}$ we choose, provides a better initial guess for the pseudospectral method.

Moreover, we show the obtained results in one trial using SympOCNet with loss $\mathcal{L}_{aug}$ and the post-process using the pseudospectral method in Figure 5.3. The left three figures show the trajectories of our proposed SympOCNet method at different times, while the right three figures correspond to the final results from the pseudospectral method at different times. In each figure, the four colored circles show the positions of the four drones at the specific time, and the curves connected to the circles show the trajectories before the current time. We observe that the shape of the trajectory given by our SympOCNet method does not change too much after the post-process, which shows that SympOCNet provides a suboptimal solution to the optimal control problem in this example.

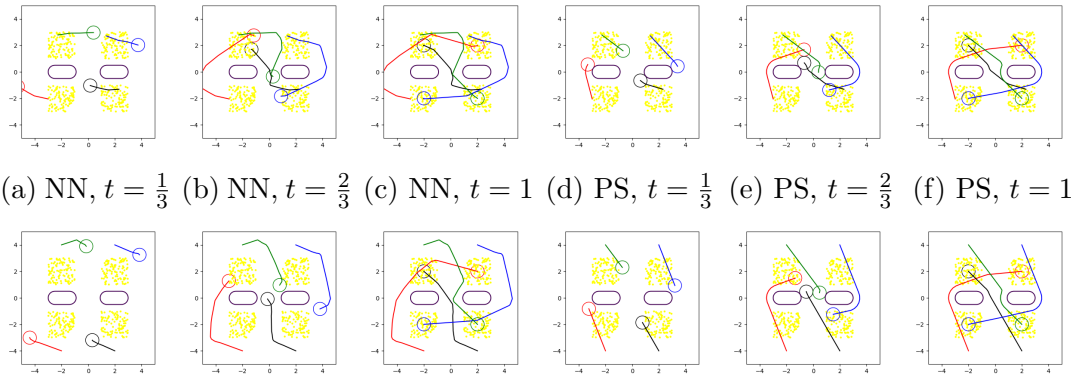
5.5.2 Generalizability and offline-online training



(a) case 1 (b) case 2 (c) case 3 (d) case 4 (e) case 5 (f) case 6

Figure 5.4: **Initial positions of drones, obstacles, and training data.** In the six figures, we show the initial positions of the four drones (represented by colored circles), the positions of the two obstacles (represented by the round cornered rectangles), and the randomly sampled training data (represented by the yellow dots) in the six testing cases in Section 5.5.2.

In this section, we consider the same problem setup as in Section 5.5.1. To test the generalizability of SympOCNet, we assume that the exact initial position is



(a) NN, $t = \frac{1}{3}$ (b) NN, $t = \frac{2}{3}$ (c) NN, $t = 1$ (d) PS, $t = \frac{1}{3}$ (e) PS, $t = \frac{2}{3}$ (f) PS, $t = 1$

(g) NN, $t = \frac{1}{3}$ (h) NN, $t = \frac{2}{3}$ (i) NN, $t = 1$ (j) PS, $t = \frac{1}{3}$ (k) PS, $t = \frac{2}{3}$ (l) PS, $t = 1$

Figure 5.5: **Output trajectories of offline-online training and post-process.** The first and second rows correspond to the first and fourth case, respectively. The left three columns are the output from L-BFGS, and the right three columns are the output from the pseudospectral method. In each figure, the four colored circles show the positions of the four drones at time t, and the curves connected to the circles show the trajectories before time t.

unknown in the training process. We train one SympNet φ_θ using multiple initial positions sampled from a uniform distribution centered at the initial position in Section 5.5.1. With multiple initial positions, we train one SympNet φ_θ and several variables $\{(\boldsymbol{y}_{0k}, \boldsymbol{u}_k, \boldsymbol{q}_{0k})\}_k$, where the k-th trainable tuple $(\boldsymbol{y}_{0k}, \boldsymbol{u}_k, \boldsymbol{q}_{0k})$ corresponds to the k-th sampled initial position. We refer to this training process of the SympNet and all trainable variables as "offline training". After $100,000$ steps of offline training, the exact initial positions are provided. To obtain an improved solution in reasonable running time using the trained SympNet φ_θ as well as the new information, we fix the SympNet φ_θ and apply the L-BFGS method to train the variables $(\boldsymbol{y}_0, \boldsymbol{u}, \boldsymbol{q}_0)$ with the exact initial and terminal positions. We call this training process using L-BFGS method the "online training". The optimal trajectory is then computed using (5.13), where $\varphi_\theta^{(1)}$ comes from the SympNet φ_θ trained in the offline training, and the parameters $(\boldsymbol{y}_0, \boldsymbol{u}, \boldsymbol{q}_0)$ are outputted from the online training. After the online training, we also apply the pseudospectral method to improve the quality of the results.

To generate training data for the initial positions, we sample 100 points from the uniform distribution in $[x-1, x+1] \times [y-1, y+1]$ where (x, y) is the initial position of a drone in Section 5.5.1. To test the generalizability of our method, we have six cases. In the first three cases, the exact initial positions are randomly sampled from the same distribution used to generate the training data. In other words, we assume the exact data is covered in the area of the training data. In the last three cases, we set the initial conditions for the state variable to be $(-2, -4, 2, -4, 2, 4, -2, 4)$, $(-4, -4, 0, -4, 0, 4, -4, 4)$, and $(0, -4, 0, -2, 0, 2, 0, 4)$, respectively. These three initial positions are not in the range of the training data. Therefore, the results with these three initial positions show the generalizability of our SympOCNet method when the test data is not covered by the area of the training data. The initial positions of the drones, the positions of the obstacles, and the training data are shown in Figure 5.4, where the four colored circles denote the drones, the two round cornered rectangles denote the obstacles, and the yellow dots denote the sampled training data.

In our experiments, we found that the performance is improved if we increase the dimension of the problem by adding some latent variables. We increase the dimension of the state space from 8 to 16, and define the Hamiltonian on the larger space by

$$H(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{p}_1, \boldsymbol{p}_2) = H_{\epsilon,a}(\boldsymbol{x}_1, \boldsymbol{p}_1) + \frac{1}{2}\|\boldsymbol{p}_2\|^2 \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{R}^8,$$

where $(\boldsymbol{x}_1, \boldsymbol{p}_1)$ are the variables in the original problem, $(\boldsymbol{x}_2, \boldsymbol{p}_2)$ are the latent variables we add, and $H_{\epsilon,a}$ is the function defined in (5.18). The minimal constraint, cost in the optimal control problem, and running time of the L-BFGS method and pseudospectral method are shown in Table 5.3. As in Section 5.5.1, we run 10 repeated experiments and put their average values in the table. In the online training, we run

1,000 L-BFGS iterations to train the parameters of all the six cases simultaneously. Note that we can train them all at once, since they share the same SympNet, and the loss function is the summation of the loss functions in all cases. Then, we run the pseudospectral method on each case until it converges. The statistics for the L-BFGS training and the pseudospectral method for the six cases are shown in each line of Table 5.3. The minimal constraint value of L-BFGS method is not as good as the results in Section 5.5.1, which is expected, since the initial positions are not known when the SympNet is trained. However, from the minimal constraint values of the pseudospectral post-process in the six cases, we conclude that our offline-online training still provides a good initialization for the pseudospectral method, which shows a good generalizability of our proposed method.

	minimal constraint	cost	time (s)
L-BFGS	-1.27E-04	231.44	158
PS (case 1)	-1.97E-11	102.00	59
PS (case 2)	-1.74E-11	94.59	77
PS (case 3)	-3.36E-11	90.95	79
PS (case 4)	-6.00E-09	141.22	65
PS (case 5)	-7.19E-08	140.47	111
PS (case 6)	1.64E-13	112.41	67

Table 5.3: **The results given by the offline-online training (L-BFGS) and the post-process (pseudospectral method) in six generalizability tests.** The offline-online training using L-BFGS method provides a good initialization for the pseudospectral method. With this initialization, the pseudospectral method in the post-process improves the feasibility and optimality of the solution in all the six cases.

We also compare our results with the pseudospectral method whose initialization is given by the linear interpolation (as descibed in Section 5.5.1) in the six generalizability tests. The minimal constraint value, cost value in the optimal control problem, and running time of the pseudospectral method with the linear initialization are shown in Table 5.4. Comparing Table 5.3 with Table 5.4, we see that although our proposed method has slightly larger cost values in the first five cases,

	minimal constraint	cost	time (s)
PS (linear initialization, case 1)	-7.71E-09	85.19	134
PS (linear initialization, case 2)	2.95E-07	83.98	173
PS (linear initialization, case 3)	-1.94E-10	79.67	178
PS (linear initialization, case 4)	-5.55E-16	80.44	157
PS (linear initialization, case 5)	-8.08E-14	136.16	147
PS (linear initialization, case 6)	-1.00	60.00	28

Table 5.4: **The results given by the pseudospectral method with the linear initialization in six generalizability tests.** Although the pseudospectral method with the linear initialization performs slightly better than our proposed method in the first five cases, it violates the constraint in the last case. Therefore, our proposed method provides a more robust initialization for the pseudospectral method compared with the linear initialization.

the pseudospectral method with the linear initialization does not converge to a feasible solution in the last case. Therefore, our proposed method provides a more robust initialization for the pseudospectral method compared with the linear initialization.

Moreover, we plot the output trajectories of one trial of the first and fourth cases computed using our proposed method in Figure 5.5. The left three columns show the output trajectory given by the offline-online training, and the right three columns are the output trajectory from the post-process. In each figure, the yellow dots are the initial positions of the sampled training data. Similar as in Section 5.5.1, the four colored circles show the current positions of the four drones, and the curves connected to them illustrate the trajectories before the current time. The first row in Figure 5.5 is for the first case where the exact initial positions are in the area of the sampled training data. The second row is for the fourth case where the exact initial positions are outside the area of the training data. We observe that the offline-online training process still provides feasible trajectories in both cases, based on which the post-process improves the smoothness of the trajectories to provide more optimal solutions. Therefore, our proposed SympOCNet method has the generalizability to handle unseen data which are close to the training data.

5.5.3 Multiple drones in a two-dimensional room

In this section, we consider the path planning problem for multiple drones in a room, which is inspired by [165]. To be specific, we consider M drones moving in a room $[-C_r, C_r]^2$ for some $C_r > 0$. In our experiments, we set the constant $C_r = 5$ (i.e., the room has size 10×10). We need to avoid the collisions among the drones as well as the collisions between drones and the walls. The constraint function $h = (h_1, h_2)$ contains two parts: the function h_2 is defined by (5.27), and the function h_1 is the obstacle constraint provided by the four walls. Hence, we set h_1 in the form of (5.25), where the function d is defined by $d(\boldsymbol{x}) = (x_1+C_r, C_r-x_1, x_2+C_r, C_r-x_2)$ for any $\boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$. The initial positions of the drones are near the boundary of the room, and the terminal positions are the opposite locations, i.e., we set $\boldsymbol{x}_T = -\boldsymbol{x}_0$. This is a high-dimensional example (the dimension of the state space is $n = 2M$, which ranges from 64 to 512 in our experiments), and hence we apply our SympOCNet method without the post-process.

First, we set the drone radius C_d to be 0.3. We run 10 repeated experiments (with $100,000$ training iterations in each experiment) with drone numbers $M = 32$ and 64 to test the robustness of our SympOCNet method. The results are shown in Table 5.5. In each trial, we compute the minimal normalized distance of any pair of drones at any time grid, i.e., we compute $\mathcal{D}$ defined by

$$\mathcal{D} = \frac{1}{2C_d} \min_{1\le i<j\le M} \min_{1\le k\le N} \|\boldsymbol{x}_i(t_k) - \boldsymbol{x}_j(t_k)\|,$$

where $\boldsymbol{x}_i(t_k)$ denotes the center position for the i-th drone at time t_k. Then, we compute the average and standard deviation of these minimal normalized distances $\mathcal{D}$ in the 10 trials, which are shown in the second and third columns in Table 5.5. We also compute the scaled cost in the optimal control problem, which is defined

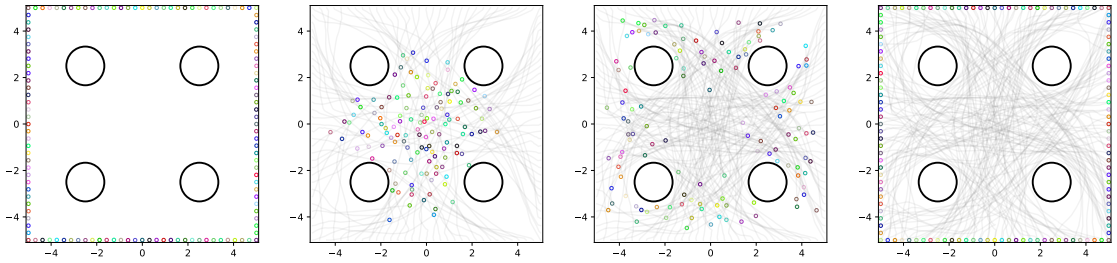
by $\frac{1}{M}\int_0^T \frac{1}{2}\|\dot{\boldsymbol{x}}(t)\|^2 dt$, where $\boldsymbol{x}(\cdot)$ is the output trajectory for the state variable $\boldsymbol{x}$ in the optimal control problem. The average of the scaled cost is shown in the fourth column in Table 5.5. In the last column of Table 5.5, we show the averaged running time of the training process. From the results, we observe that our SympOCNet method provides feasible results in reasonable running time for this high-dimensional problem. Note that the collision between two drones happens whenever the minimal normalized distance is less than 1. Therefore, in general the collision does not happen for 32 drones, but it may happen for 64 or more drones. To provide a more feasible solution, in the following experiments, we put a safe zone outside each drone by setting C_d to be a little bit larger than the actual drone radius.

# drones	$\mathbb{E}(\mathcal{D})$	std($\mathcal{D}$)	scaled cost	time (s)
32	1.001	0.00390	85.84	1905
64	0.986	0.00347	100.43	2214

Table 5.5: **The results for the path planning problem with** 32 **and** 64 **drones in a room.** We show the expected value and standard deviation of the minimal normalized distances. The collision does not happen for 32 drones, but it may happen for 64 or more drones. Therefore, we need to put a safe zone near each drone to avoid collision when the number of drones is large. The computation time increases only by 309 seconds when scaling from 32 to 64 drones.

We also tested our SympOCNet method on this path planning problem with 128 drones (whose radius is 0.075) and four obstacles. The initial and terminal positions are similar to the cases of 32 or 64 drones (see Figure 5.6 (a) and (d)). The four obstacles are balls inside the room, which are represented by the black circles in Figure 5.6. The constraint function d_j in (5.26) corresponding to the obstacle E_j is defined by $d_j(\boldsymbol{x}) = \|\boldsymbol{z}_j - \boldsymbol{x}\|^2 - (C_o + C_d)^2$ for any $\boldsymbol{x} \in \mathbb{R}^n$, where $\boldsymbol{z}_j \in \mathbb{R}^2$ and $C_o > 0$ are the center and radius of the obstacle E_j, respectively. The results are shown in Figure 5.6. Each figure shows the current positions (represented by colored circles) and the previous trajectories (represented by grey curves) of the drones at different time $t = 0, \frac{1}{3}, \frac{2}{3}, 1$. To provide a feasible solution, we set C_d in the constraint

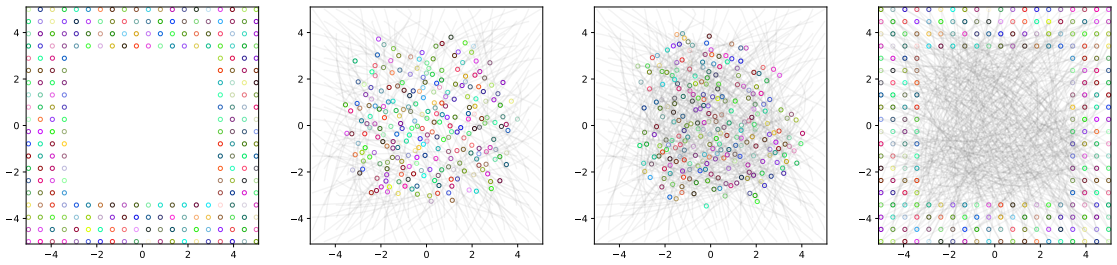
function to be 0.1 instead of 0.075. It takes 4161 seconds to finish $100,000$ training iterations. From the numerical results, we found that there are no collisions between obstacles and drones, and hence SympOCNet performs well and efficiently in this high-dimensional problem.



(a) $t = 0$ (b) $t = \frac{1}{3}$ (c) $t = \frac{2}{3}$ (d) $t = 1$

Figure 5.6: **Path planning of 128 drones with four obstacles on the plane.** We plot the predicted positions of 128 drones at time $t = 0, \frac{1}{3}, \frac{2}{3}, 1$. The four black circles represent the four obstacles, and the colored circles represent the drones. The paths from the initial conditions to the current positions of all drones are plotted as gray lines. There are no collisions in the three clipped snapshots, and the planned trajectories are all inside the 10×10 square.

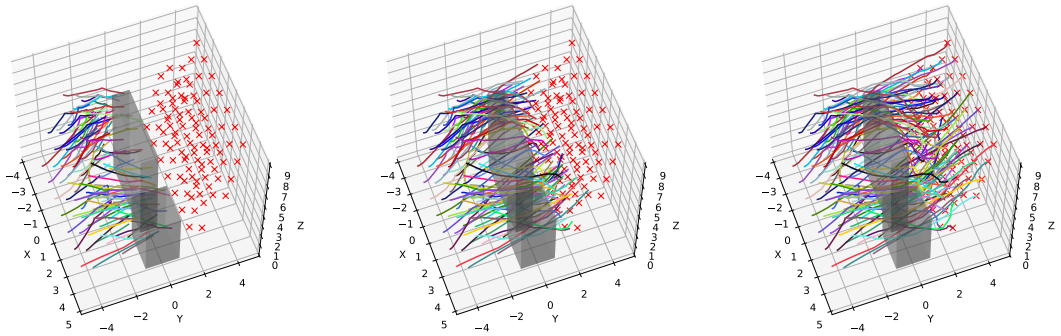
Then, we tested our proposed method on a higher dimensional problem. We set the drone number $M = 256$, and hence the dimension of the state space is $n = 2M = 512$. We solve this path planning problem with 256 drones (whose radius is 0.09) in the two-dimensional room $[-5, 5]^2$. The initial positions are shown in Figure 5.7 (a), and the terminal condition is $\boldsymbol{x}_T = -\boldsymbol{x}_0$. The results are shown in Figure 5.7. To provide a feasible solution, we add a safe zone and set C_d in the state constraint to be 0.1 instead of 0.09. From this result, we observe no collisions among the drones. It takes 5481 seconds to obtain this feasible solution in $100,000$ training iterations. Therefore, SympOCNet is able to efficiently solve this 512-dimensional path planning problem.



(a) $t = 0$ (b) $t = \frac{1}{3}$ (c) $t = \frac{2}{3}$ (d) $t = 1$

Figure 5.7: **Path planning of 256 drones on the plane.** We plot the predicted positions of 256 drones at time $t = 0, \frac{1}{3}, \frac{2}{3}, 1$. The drones are represented by colored circles. The paths from the initial positions to the current positions of all drones are plotted as gray lines. There are no collisions in the three clipped snapshots, and the planned trajectories are inside the 10×10 square.

5.5.4 Multiple drones with obstacle avoidance in a three-dimensional space



(a) $t = \frac{1}{3}$ (b) $t = \frac{2}{3}$ (c) $t = 1$

Figure 5.8: **Path planning of 100 drones in a 3d space.** We plot the predicted positions of 100 drones at time $t = \frac{1}{3}, \frac{2}{3}, 1$. The paths from the initial positions to the current positions of all drones are plotted as colored lines. The destinations are marked as red crosses. The drones reach their destinations without any collision.

We consider the three-dimensional swarm path planning example in [148]. To be specific, we consider $M = 100$ drones with radius 0.18. Their goal is to avoid two obstacles which are located between their initial positions and terminal positions, as illustrated in Figure 5.8. The total dimension of the state space is $n = 3M = 300$. The j-th obstacle E_j is defined to be the rectangular set $[C_{11}^j, C_{12}^j] \times [C_{21}^j, C_{22}^j] \times$

$[C_{31}^j, C_{32}^j]$, where C_{ik}^j is a constant scalar. We set the constraint function $h = (h_1, h_2)$ in the same way as described before, where the function d_j in (5.26) is defined by

$$d_j(\boldsymbol{x}) = \max\{C_{i1}^j - C_d - x_i, x_i - C_{i2}^j - C_d \colon i = 1, 2, 3\},$$

for each $j = 1, 2$ and $\boldsymbol{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$. Note that $d_j(\boldsymbol{x}) \leq 0$ if and only if $x_i \in [C_{i1}^j - C_d, C_{i2}^j + C_d]$ for each $i = 1, 2, 3$, which holds if the ball centered at $\boldsymbol{x}$ with radius C_d intersects with the obstacle E_j. Therefore, the constraint $d_j(\boldsymbol{x}) \geq 0$ enforces the drones to avoid collisions with the obstacle E_j. In our experiment, we set

$$(C_{11}^1, C_{12}^1, C_{21}^1, C_{22}^1, C_{31}^1, C_{32}^1) = (-1.8, 1.8, -0.3, 0.3, 0.2, 6.8),$$
$$(C_{11}^2, C_{12}^2, C_{21}^2, C_{22}^2, C_{31}^2, C_{32}^2) = (2.2, 3.8, -0.8, 0.8, 0.2, 3.8).$$

In other words, the constraint sets are $[-1.8, 1.8] \times [-0.3, 0.3] \times [0.2, 6.8]$ and $[2.2, 3.8] \times [-0.8, 0.8] \times [0.2, 3.8]$. We apply our proposed SympOCNet method to solve this 300-dimensional problem, and plot the result in Figure 5.8. Due to the high dimensionality, we do not apply the post-process. To provide a feasible solution, we set C_d in the state constraint to be 0.2 instead of 0.18. From the numerical results, we observe no collisions among the drones and the obstacles. The minimal distance between every pair of drones is 0.3994, which is bigger than twice of the radius. It takes 3633 seconds to finish $150,000$ training iterations. Therefore, our proposed SympOCNet method provides a feasible solution to this high-dimensional swarm path planning problem in reasonable time.

5.6 Summary

We have proposed a novel SympOCNet method for efficiently solving high-dimensional optimal control problems with state constraints. We applied SympOCNet to several multi-agent simultaneous path planning problems with obstacle avoidance. The numerical results show SympOCNet's ability to solve high-dimensional problems efficiently with dimension more than 500. These first results reveal the potential of SympOCNet for solving high-dimensional optimal control problems in real-time.

One of the key modeling assumptions of SympOCNet is that there exists a symplectic transformation that renders the Hamiltonian associated with the optimal control problem state independent. In general, whether there exists such a symplectic transformation remains unknown theoretically. We will investigate the class of functions $\mathcal{F}$ that could be represented by the symplectic transformation of an affine map in future works. Empirically, we observed reasonably accurate results in all the experiments shown in the chapter, which justifies that $\mathcal{F}$ should be large enough to approximate the phase flow of Hamiltonian systems arising from optimal control problems with quadratic cost and linear integrators. We will consider more complicated cost structures and dynamics in the future, which may require more expressivity of the latent trajectory and the coordinate transformation. Our preliminary results show that by parameterizing the coordinate transformation through a time-dependent symplectic map, we can solve problems with more complicated dynamics, such as the double-integrator problems.

We are also going to consider possible combinations with the DeepONet architecture [130] to avoid any training cost during inference and to endow the predictive system with uncertainties, such as uncertain initial and terminal positions in path

planning problems.

Chapter Six

TSympOCNet for nonconvex path planning problems with linear and nonlinear dynamics

6.1 Introduction

Recent advances in neural networks show promise in overcoming the curse of dimensionality by encoding physical information into network architectures and loss functions [158]. Leveraging this, SympOCNet [141] was proposed to solve the Hamilton's equation corresponding to the high-dimensional optimal control problem. SympOCNet consists of two parts: (i) a latent representation which are parameterized by affine maps and (ii) a coordinate transformation module which are parameterized by symplectic networks (SympNets) [102]. Even though SympOCNet demonstrates the scalability in high dimensions, e.g., being able to plan the path for 256 agents, its formulation is mainly restricted to the linear integrator, i.e., controlling the velocity of vehicles. In this work, we extend the framework and introduce a novel architecture named TSympOCNet to handle more general dynamics and control problems. This is achieved by introducing a time-dependent module in the coordinate transformation and parameterize the latent space by solution to a particular linear quadratic regulator (LQR) problem. One of the main differences between this work and SympOCNet is that we keep the dimension of the physical space and the latent space the same.

The rest of the chapter is organized as follows: Section 6.2 provides an overview of the background information to be utilized, covering optimal control problem setup (Section 6.2.1) and SympOCNets (Section 6.2.2). In Section 6.3, we introduce our problem setup (Section 6.3.1.1) and the TSympOCNet architecture (Section 6.3.2) which consists of a latent representation and a coordinate transformation. The comprehensive training algorithm is elaborated in Section 6.3.3. In Section 6.4, we apply our method to multi-agent path planning problems with obstacle and collision avoidance. Simulations with single and four agents demonstrate robustness and ef-

fectiveness, validated against the shooting method in Section 6.4.1. Sections 6.4.2 and 6.4.3 showcase the effectiveness and efficiency of TSympOCNet on handling high-dimensional problems and problems with complicated constraints. Finally, a synthesis of findings and concluding remarks are presented in Section 6.5.

6.2 Preliminary background

This section provides some background materials used in the remainder of the chapter. In Section 6.2.1, we briefly recall the optimal control problem, the corresponding Hamiltonian system, and shooting method. Section 6.2.2 is a review of the SympOCNet architectures, which is a first attempt by the authors to employ neural network architecture for solving optimal control problems.

6.2.1 Optimal Control Problem, Hamiltonian ODE system and shooting method

Let us consider a finite horizon deterministic optimal control problem

$$\inf_{\boldsymbol{u}(\cdot)\in\mathcal{U}} \left\{ \int_0^T L(\boldsymbol{x}(s), \boldsymbol{u}(s)) ds \right\} , \tag{6.1a}$$

over the set of trajectories $(\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot))$ satisfying

$$\begin{cases} \dot{\boldsymbol{x}}(s) = f(\boldsymbol{x}(s), \boldsymbol{u}(s)), \quad \forall\ s \in [0, T] , \\ \boldsymbol{x}(0) = \boldsymbol{x}^{(0)},\ \boldsymbol{x}(T) = \boldsymbol{x}^{(T)} . \end{cases} \tag{6.1b}$$

Here, $\mathcal{U} := \{\boldsymbol{u} : [0,T] \to U \subset \mathbb{R}^m \mid \boldsymbol{u}(\cdot) \text{ is measurable}\}$ is the set of controls. The Lagrangian (or running cost) $L : \mathbb{R}^n \times U \to \mathbb{R}$, the dynamics $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ are given functions, with basic regularity properties: bounded, continuous and Lipschitz continuous w.r.t all variables.

A necessary optimality condition for the problem (6.1) is given by the Pontryagin's maximum principle (see for instance [186]). By introducing the so-called costate $\boldsymbol{p} : [0,T] \to \mathbb{R}^n$, the optimal trajectory $\boldsymbol{x}$ of the control problem together with the costate $\boldsymbol{p}$ satisfy the following Hamiltonian ODE system

$$\begin{cases} \dot{\boldsymbol{x}}(s) = \nabla_{\boldsymbol{p}} H(\boldsymbol{x}(s), \boldsymbol{p}(s)) \ , \\ \dot{\boldsymbol{p}}(s) = -\nabla_{\boldsymbol{x}} H(\boldsymbol{x}(s), \boldsymbol{p}(s)) \ , \end{cases} \tag{6.2a}$$

for every $s \in [0,T]$, with initial, final condition

$$\boldsymbol{x}(0) = \boldsymbol{x}^{(0)}, \ \boldsymbol{x}(T) = \boldsymbol{x}^{(T)} \ , \tag{6.2b}$$

where the Hamiltonian $H \colon \mathbb{R}^{2n} \to \mathbb{R}$ is defined by

$$H(\boldsymbol{x}, \boldsymbol{p}) = \max_{\boldsymbol{u} \in U} \{ \langle \boldsymbol{p}, f(\boldsymbol{x}, \boldsymbol{u}) \rangle - L(\boldsymbol{x}, \boldsymbol{u}) \} \ . \tag{6.2c}$$

One classical numerical method to solve system (6.2) is the shooting method, which converts the initial-terminal system to a root finding problem (see for instance [154]). More detailed, let us consider the same Hamltonian system (6.2) but with (6.2b) replaced by a initial condition

$$\boldsymbol{x}(0) = \boldsymbol{x}^{(0)}, \ \boldsymbol{p}(0) = \boldsymbol{p}^{(0)} \ . \tag{6.3}$$

For a given $\boldsymbol{p}^{(0)}$, solving this system gives a trajectory together with the co-state $(\boldsymbol{x}(s), \boldsymbol{p}(s))$, for every $s \in [0, T]$. We can then define a shooting function S maps $\boldsymbol{p}^{(0)}$ to the value of the final conditions, for instance $S(p^{(0)}) = \|x(T) - x^{(T)}\|$. Finding a zero of S gives a solution to (6.2). This is typically done by (quasi-)Newton method. However, this method is significant sensitive to the initial guess. The algorithm may either not converge at all, or converge to a local minimum in the presence of nonconvexity.

6.2.2 SympOCNets

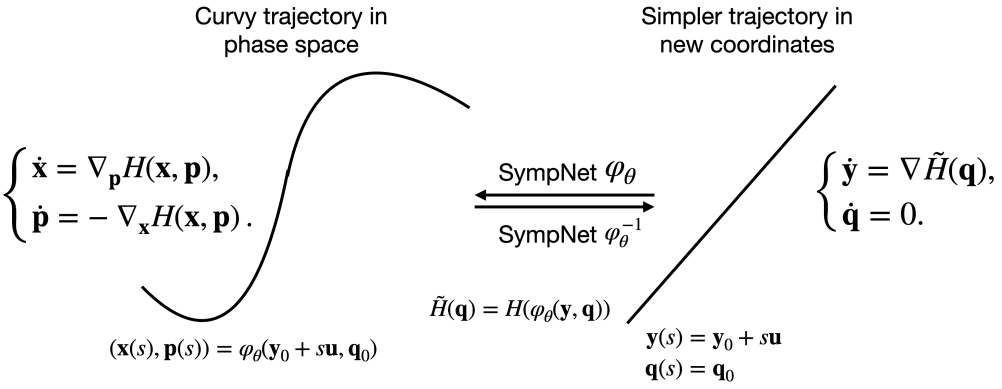


Figure 6.1: **An illustration of the SympOCNet method.** SympOCNet φ_θ maps curvy trajectory in the phase space to affine maps in new coordinates.

SympOCNet was introduced in [141] to first solve the optimal control problem (6.1) together with the system (6.2), and then extended to include a state constrained case. It focus on a special case of dynamics

$$\dot{\boldsymbol{x}}(s) = \boldsymbol{u}(s), \ \forall \ s \in [0, T] \ . \tag{6.4}$$

The crucial step of SympOCNet is employing a change of variables technique in the phase space, by a symplectic map represented by SympNet architecture, to transform the original Hamiltonian ODE system (6.2) into a simper one that can be easily

solved. More detailed, assuming given a symlectic map $\phi : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$, the change of variables is defined by $(\boldsymbol{y}, \boldsymbol{q}) = \phi(\boldsymbol{x}, \boldsymbol{p})$. Then, if $(\boldsymbol{x}(s), \boldsymbol{p}(s))$, for $s \in [0, T]$, is a solution of system (6.2), we assume $(\boldsymbol{y}(s), \boldsymbol{q}(s))$, for $s \in [0, T]$, is a solution of a new Hamiltonian ODE system, with the Hamiltonian taking the form

$$\tilde{H}(y, q) = H(\phi^{-1}(y, q)) \ . \tag{6.5}$$

For computational efficiency, it is also assumed in [141] that the new Hamiltonian $\tilde{H}$ does not dependent on $\boldsymbol{y}$, leading to a system:

$$\begin{cases} \dot{\boldsymbol{y}}(s) = \nabla_{\boldsymbol{p}} \tilde{H}(\boldsymbol{q}(s)) \ , \\ \dot{\boldsymbol{q}}(s) = 0 \ , \end{cases} \tag{6.6a}$$

for every $s \in [0, T]$, with initial, final condition

$$(\phi^{-1})^{(1)}(\boldsymbol{y}(0), \boldsymbol{q}(0)) = \boldsymbol{x}^{(0)}, \ (\phi^{-1})^{(1)}(\boldsymbol{y}(T), \boldsymbol{q}(T)) = \boldsymbol{x}^{(T)} \ , \tag{6.6b}$$

where $f^{(1)}$ denotes the first n output components of any $f : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$. The inverse of this symplectic map ϕ^{-1} is then approximated through a parameterized family of symplectic maps φ_θ, where θ represents the unknown parameters to be learned through SympNet architecture. The solution to the original problem can be obtained by mapping the trajectory back to original phase space through $\varphi_\theta = \phi^{-1}$. We refer to [141] for more details.

6.3 TSympOCNet for optimal control problems with state constraints

In this section, we purpose to solve optimal control problems with time-dependent symplectic optimal control network (TSympOCNet), a neural network parameterized by an LQR latent representation and a time-dependent symplectic coordinate transformation.

6.3.1 Control Problem and A Sketch of the Architecture

6.3.1.1 Problem Setup

We are interested in the optimal control problem (6.1) with more general, potentially nonlinear with respect to x, dynamics

$$\dot{\boldsymbol{x}}(s) = f(\boldsymbol{x}(s)) + B\boldsymbol{u}(s),\ \forall\ s \in [0, T]\ . \tag{6.7}$$

Moreover, we consider the state constraint case, that is in the constraint (6.1), we further require

$$h(\boldsymbol{x}(s)) \geq 0,\ \forall\ s \in [0, T]\ . \tag{6.8}$$

Here, $h : \mathbb{R}^n \to \mathbb{R}^{n'}$ is a function imposing the constraints on the state variable $\boldsymbol{x}$. For instance, to avoid obstacles, h can be defined using the signed distance function to the obstacles. We adapt a similar technique, the soft penalty method, as in [141] to convert the constrained problem into an unconstrained one. Consider a penalty function $U_{\epsilon,l} : \mathbb{R}^{n'} \to \mathbb{R}$, dependent on positive hyperparameters ϵ and l, and is

defined as follows: for every $\mathrm{h} = (\mathrm{h}_1, \ldots, \mathrm{h}_{n'}) \in \mathbb{R}^{n'}$

$$U_{\epsilon,l}(\mathrm{h}) := \max_{i \in \{1,\cdots,n'\}} U^i_{\epsilon,l}(\mathrm{h}_i) \ , \tag{6.9a}$$

where for every $i \in \{1, \ldots, m\}$, $U^i_{\epsilon,l} : \mathbb{R} \to \mathbb{R}$ is defined by

$$U^i_{\epsilon,l}(\mathrm{h}_i) := \begin{cases} -\epsilon \log(\mathrm{h}_i), & \text{if } \mathrm{h}_i > l \ , \\ -\epsilon \log(\mathrm{h}_i) + \dfrac{\epsilon}{2}\left(\left(\dfrac{\mathrm{h}_i - 2l}{l}\right)^2 - 1\right), & \text{if } \mathrm{h}_i \leq l \ . \end{cases} \tag{6.9b}$$

With this penalty function, the state constrained problem is converted to an unconstrained one with Lagrangian (or running cost)

$$L(\boldsymbol{x}(s), \boldsymbol{u}(s)) + U_{\epsilon,l}(h(\boldsymbol{x}(s)) \ . \tag{6.10}$$

Although the method we proposed in the present chapter is applicable in a wider range of problems, we are particularly interested in the context of path planning problems involving obstacle and collision avoidance. We assume the control system consists of M elementary dynamical subsystems in interaction. The state of each subsystem has a dimension d_x, where the concatenated state variable $\boldsymbol{x} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M) \in \mathbb{R}^{Md_x}$. Similarly, the control vector $\boldsymbol{u} = (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_M) \in \mathbb{R}^{Md_u}$. For each subsystem i, denote by $F_i : \mathbb{R}^{d_x} \to \mathbb{R}$ an individual "potential energy" function associated with state x_i, and $G_i : \mathbb{R}^{d_u} \to \mathbb{R}$ a convex individual "kinetic energy" function associated with control u_i. The individual dynamics of the subsystem have the form $\dot{x}_i(s) = f_i(x_i(s)) + B_i u_i(s)$, for every $s \in [0, T]$. Moreover, h will represent both the interaction (i.e., collision avoidance) and the state constraint (i.e., obstacle) of the problem. We look for a trajectory $x(s)$, for $s \in [0, T]$, minimizing the total action functional. This can be interpreted using the framework of optimal control

problem (6.1) by taking

$$\begin{aligned} L(\boldsymbol{x},\boldsymbol{u}) &= \sum_{i=1}^{M}\Big(F_i(\boldsymbol{x}_i)+G_i(\boldsymbol{u}_i)\Big)\ , \\ \dot{\boldsymbol{x}}_i(s) &= f_i(\boldsymbol{x}_i(s))+B_i\boldsymbol{u}_i(s),\ \forall\ s\in[0,T] \text{ and } \ i\in\{1,\cdots,M\}\ , \end{aligned} \tag{6.11}$$

together with state constraint (6.8). Observe that the Lagrangian (or running cost) appearing in (6.11) is the sum of kinetic and potential energy, instead of their difference as in classical mechanics. Lagrangians of the form (6.11), in which the potential is typically coercive (tending to ∞ as $\|x\|\to\infty$), are the most natural ones in the context of optimal control. In particular, thanks to coercivity of the potential, the minimization problem is well defined over arbitrary time horizon.

Combing the techniques of converting the state constraints to running cost as in (6.10), we have the Hamiltonain of the problem takes the form

$$\begin{aligned} H_{\epsilon,l}(\boldsymbol{x},\boldsymbol{p}) &= \max_{\boldsymbol{u}}\left\{\langle\boldsymbol{p},f(\boldsymbol{x})+B\boldsymbol{u}\rangle - L(\boldsymbol{x},\boldsymbol{u}) - U_{\epsilon,l}(h(\boldsymbol{x}))\right\} \\ &= \sum_{i=1}^{M}\Big(\langle\boldsymbol{p}_i,f_i(\boldsymbol{x}_i)\rangle - F_i(\boldsymbol{x}_i)+G_i^*(B_i^\top\boldsymbol{p}_i)\Big) - U_{\epsilon,l}(h(\boldsymbol{x}))\ , \end{aligned} \tag{6.12}$$

where G_i^* denotes the Legendre-Fenchel transform of G_i.

6.3.1.2 Motivation for the New Architecture

In SympOCNet architecture, for computational efficiency, we assume that $\tilde{H}$ does not depend on $\boldsymbol{y}$ in (6.6), leading to simpler affine maps trajectories in the new coordinates (see Figure 6.1). However, the existence of such a symplectic map that transforms (6.2) to (6.6) is quite mysterious and not guaranteed, in particular since ϕ is itself an isomorphism.

In the present chapter, we aim to handle more general dynamics of the form (6.7), then it is natural to consider a more expressive Hamiltonian system in the latent space. So we consider a simpler Hamiltonian ODE system in the latent space, but with Hamiltonian $\tilde{H}$ depends both on the state and co-state. Moreover, for a Hamiltonian system, we denote Φ_t the map sending any initial condition z_0 to $z(t)$, for a fixed t. Recall that if H is of class $\mathcal{C}^2$, Φ_t is a symplectic transformation (see for instance [84]).

Based on these observations, it is tempting to assume that, by parameterizing the coordinate transformation through a time-dependent symplectic map, we can transform the original Hamiltonian ODE system into a simper one. Then, the solution of the original system at a fixed time t is the preimage of the solution in the latent system by this time-dependent symplectic map. These will be detailed in the following sections.

6.3.2 TSympOCNet architecture

6.3.2.1 Latent representation

In the latent space, we consider a simper Hamiltonian ODE system, with Hamiltonian

$$\tilde{H}(\boldsymbol{y},\boldsymbol{q}) = \sum_{i=1}^{M} \left(\langle \boldsymbol{q}_i, A_i \boldsymbol{y}_i \rangle - \langle \boldsymbol{y}_i, Q_i \boldsymbol{y}_i \rangle + \langle B_i^\top \boldsymbol{q}_i, R_i B_i^\top \boldsymbol{q}_i \rangle \right) , \tag{6.13a}$$

where $A_i = \mathbf{J} f_i(0)$, $Q_i = \frac{1}{2}\mathbf{H}(F_i)(0)$, $R_i = \frac{1}{2}\mathbf{H}(G_i^*)(0)$. Here $\mathbf{J} f_i(0)$ represents the Jacobian of f_i at 0 and $\mathbf{H}(F_i)(0)$ represents the Hessian of F_i at 0. The Hamiltonian

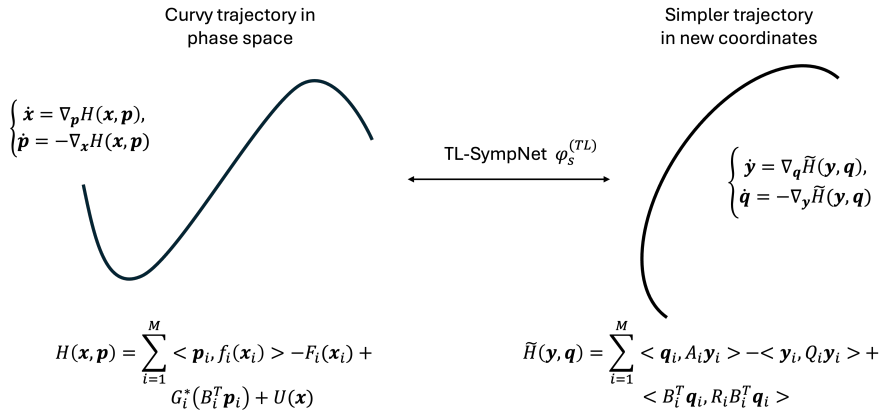


Figure 6.2: **An illustration of the time-dependent SympOCNet method.** We propose to use a time-dependent SympNet φ_s to map curvy trajectory in the phase space to simpler trajectory in new coordinates and solve the corresponding Hamiltonian system of the optimal control problem.

system then has the form

$$\begin{cases} \dot{\boldsymbol{y}}_i(s) = \nabla_{\boldsymbol{q}_i} \tilde{H}(\boldsymbol{y}(s), \boldsymbol{q}(s)), & i \in \{1, \dots, M\} \ , \\ \dot{\boldsymbol{q}}_i(s) = -\nabla_{\boldsymbol{y}_i} \tilde{H}(\boldsymbol{y}(s), \boldsymbol{q}(s)), & i \in \{1, \dots, M\} \ , \end{cases} \tag{6.13b}$$

for every $s \in [0, T]$. Moreover, we set the same initial, final condition as the orginal Hamiltonian ODE system. I.e.,

$$y(0) = x^{(0)}, \ y(T) = x^{(T)} \ . \tag{6.13c}$$

Notice that there is a one-to-one correspondence between the latent Hamiltonian system (6.13) and the following linear quadratic regulator (LQR) problem

$$\begin{aligned}&\inf_{\boldsymbol{u}(\cdot)} \int_0^T \sum_{i=1}^M \left(\boldsymbol{y}_i(s)^\top Q_i \boldsymbol{y}_i(s) + \boldsymbol{u}_i(s)^\top R_i \boldsymbol{u}_i(s)\right) ds \\ &\text{s.t.} \begin{cases} \dot{\boldsymbol{y}}_i(s) = A_i \boldsymbol{y}_i(s) + B_i \boldsymbol{u}_i(s), \quad \forall i \in \{1, \cdots, M\} \text{ and } \forall s \in [0, T] \ , \\ \boldsymbol{y}(0) = \boldsymbol{x}^{(0)}, \ \boldsymbol{y}(T) = \boldsymbol{x}^{(T)} \ . \end{cases}\end{aligned} \tag{6.14}$$

The Hamiltonian system (6.13) can be formulated as

$$\left\{ \frac{d}{ds} \begin{pmatrix} \boldsymbol{y}_i(s) \\ \boldsymbol{q}_i(s) \end{pmatrix} = \begin{pmatrix} A_i & -B_i R_i^{-1} B_i^\top \\ -Q_i & -A_i^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{y}_i(s) \\ \boldsymbol{q}_i(s) \end{pmatrix}, \ i \in \{1, \cdots, M\} \ , \right. \tag{6.15}$$

for every $s \in [0, T]$, with initial, final condition (6.13c). We call the dynamic matrix in (6.15) the Hamiltonian matrix, and denote as H_i. The solution of this two-point boundary value system can be explicitly obtained. Notice that $(\boldsymbol{y}(s), \boldsymbol{q}(s)) = ((\boldsymbol{y}_i)(s), \boldsymbol{q}_i(s))_{i=\{1,\ldots,M\}}$, where

$$\begin{pmatrix} \boldsymbol{y}_i(s) \\ \boldsymbol{q}_i(s) \end{pmatrix} = e^{H_i s} \begin{pmatrix} \boldsymbol{x}_i^{(0)} \\ \boldsymbol{q}_i^{(0)} \end{pmatrix} \ , \tag{6.16}$$

solves the system (6.15) with initial condition $\boldsymbol{y}(0) = \boldsymbol{x}^{(0)}, \boldsymbol{q}(0) = \boldsymbol{q}^{(0)}$. So if we can find an appropriate $\boldsymbol{q}_0$ such that the solution also satisfies $\boldsymbol{y}(T) = \boldsymbol{x}^{(T)}$, it is also a solution of the boundary value problem (6.15). Indeed, solution of the linear equations

$$\boldsymbol{y}_i(T) = \begin{pmatrix} I & 0 \end{pmatrix} \cdot e^{H_i T} \begin{pmatrix} \boldsymbol{x}_{0i} \\ \boldsymbol{q}_{0i} \end{pmatrix} \tag{6.17}$$

provides the desirable $\boldsymbol{q}_0$. The time complexity for this method is $O(Md_x)$.

Remark 5. The problem (6.14) can also be approximated through a finite horizon LQR problem with quadratic terminal cost

$$\begin{aligned} &\inf_{\boldsymbol{u}(\cdot)} \int_0^T \boldsymbol{y}(s)^\top Q\boldsymbol{y}(s) + \boldsymbol{u}(s)^\top R\boldsymbol{u}(s)ds + \boldsymbol{y}(T)^\top Q_f\boldsymbol{y}(T) \\ &\text{s.t.} \begin{cases} \dot{\boldsymbol{y}}(s) = A\boldsymbol{y}(s) + B\boldsymbol{u}(s), \quad \forall s \in [0,T] \ , \\ \boldsymbol{y}(0) = \boldsymbol{x} \ . \end{cases} \end{aligned} \tag{6.18}$$

Let us define the value function V which associates with any $(x,t) \in \mathbb{R}^{Md_x} \times [0,T]$ the infimum of $\int_t^T \boldsymbol{y}(s)^\top Q\boldsymbol{y}(s) + \boldsymbol{u}(s)^\top R\boldsymbol{u}(s)ds + \boldsymbol{y}(T)^\top Q_f\boldsymbol{y}(T)$, under the constraints in (6.18). Then $V(x,t)$ will remain quadratic form $x^\top P(t)x$ for every $t \in [0,T]$, with $P(t)$ the solution of the following differential Riccati equation

$$\begin{cases} \dot{P}(s) = -A^\top P(s) - P(s)A + P(s)BR^{-1}B^\top P(s) - Q, \quad s \in [t,T] \ , \\ P(T) = Q_f \ . \end{cases} \tag{6.19}$$

Notice that solving the Riccati equation, through the complexity is $O(Md_x^3)$, gives an optimal control as a function of the state $\boldsymbol{x}$. It is also called feedback control or closed loop control, which is know to have advantages in real application, for instance the solution is robust against system perturbations. Constructing a robust neural network architecture for the control problem is of independent interest, and we leave it as a direction for future work.

6.3.2.2 Coordinate transformation

In the reminder of this chapter, we will denote $(\boldsymbol{x}(s), \boldsymbol{p}(s))_{s\in[0,T]}$ the solution of the Hamiltonian system in the phase space with Hamiltonian (6.12), and $(\boldsymbol{y}(s), \boldsymbol{q}(s))_{s\in[0,T]}$ the solution of the Hamiltonian system (6.13) in the latent space.

The goal is to represent and approximate the inverse of an unknown time-dependent symplectic map φ, that can transform $(\boldsymbol{x}(t), \boldsymbol{p}(t))$ to $(\boldsymbol{y}(t), \boldsymbol{q}(t))$, for any fixed $t \in [0, T]$.

Definition 23. Let U be an open set in $\mathbb{R}^{2n}$. We call a map $\varphi \in \mathcal{C}^1(U \times [0, T]; \mathbb{R}^{2n})$ is time-dependent symplectic if for any fixed $t \in [0, T]$, $\varphi(\cdot, t)$ is symplectic. I.e., for any $t \in [0, T]$,

$$\nabla_{\boldsymbol{z}} \varphi(\boldsymbol{z}, t)^\top J \nabla_{\boldsymbol{z}} \varphi(\boldsymbol{z}, t) = J, \ \forall \ \boldsymbol{z} \in U \ . \tag{6.20}$$

For easy expression, we use φ_t to denote $\varphi(\cdot, t)$, for a fixed $t \in [0, T]$. In this chapter, we consider the *linear* and *affine* time-dependent symplectic map φ.

Remark 6. We refer to a time-dependent symplectic transformation that is linear (affine) in z as a linear (affine) time-dependent symplectic transformation. In the linear case, each φ_t is a linear symplectic transformation that can be represented by a symplectic matrix.

Lemma 7 ([97])**.** For any symplectic matrix $K \in \mathbb{R}^{2n \times 2n}$, there exists $k = 5$ symmetric matrices $\{S_i\}_{i=1}^k$, such that

$$K = \begin{pmatrix} I & 0 \\ S_1 & I \end{pmatrix} \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} \cdots \begin{pmatrix} I & 0 \\ S_k & I \end{pmatrix} . \tag{6.21}$$

Lemma 7 shows that any linear symplectic transformation can be written as the composition of at most $k = 5$ unit block lower/upper triangular matrices. Inspired by the lemma above, we propose the TL-SympNet which resembles the G-SympNet to represent arbitrary affine time-dependent symplectic transformation.

Definition 24. For $i \in \{1, 2, \ldots, N\}$, let $\tilde{\sigma}_{K^i, \boldsymbol{a}^i, \boldsymbol{b}^i}(\boldsymbol{x}; t) := (K^i)^\top (\boldsymbol{a}^i(t) \odot (K^i \boldsymbol{x} + \boldsymbol{b}^i))$ for any $\boldsymbol{x} \in \mathbb{R}^n$, $t \in [0, T]$. Any TL-SympNet $\tilde{\varphi}_t$ is an alternating composition of the

following two parameterized functions:

$$\begin{aligned}\mathcal{T}_t^{low,i}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} &= \begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}+\tilde{\sigma}_{K^i,\boldsymbol{a}^i,\boldsymbol{b}^i}(\boldsymbol{x};t)\end{pmatrix} \quad \forall \boldsymbol{x},\boldsymbol{p}\in\mathbb{R}^n,\\ \mathcal{T}_t^{up,i}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} &= \begin{pmatrix}\tilde{\sigma}_{K^i,\boldsymbol{a}^i,\boldsymbol{b}^i}(\boldsymbol{p};t)+\boldsymbol{x}\\ \boldsymbol{p}\end{pmatrix} \quad \forall \boldsymbol{x},\boldsymbol{p}\in\mathbb{R}^n,\end{aligned} \tag{6.22a}$$

such that

$$\tilde{\varphi}_t = \mathcal{T}_t^{up,N}\circ\mathcal{T}_t^{low,N}\cdots\mathcal{T}_t^{up,1}\circ\mathcal{T}_t^{low,1} \text{ or } \tilde{\varphi}_t = \mathcal{T}_t^{low,N}\circ\mathcal{T}_t^{up,N}\cdots\mathcal{T}_t^{low,1}\circ\mathcal{T}_t^{up,1}, \tag{6.22b}$$

where the learnable parameters are from the matrices $K^i\in\mathbb{R}^{l\times n}$, the vectors $\boldsymbol{b}^i\in\mathbb{R}^l$, and fully connected neural network $\boldsymbol{a}^i:\mathbb{R}\to\mathbb{R}^l$. The dimension l (which is the dimension of $\boldsymbol{b}^i$ as well as the number of rows in K^i) is a positive integer that can be tuned, called the width of TL-SympNet. N is the number of layers of TL-SympNet. We refer to the number of layers and width of each $\boldsymbol{a}^i$ as the sublayers and subwidth of TL-SympNet.

Then, following directly from the universal approximation theorem of standard neural networks, we have the following result:

Theorem 16. The map $\tilde{\varphi}_t$ defined in (6.22) is an universal approximator within the set of affine time-dependent symplectic transformations.

Recall that the original Hamiltonian system and the Hamiltonian system (6.13) in latent space share the same set of boundary values $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(T)}$. This inspires us to design a neural network architecture which preserves the boundary values of $\boldsymbol{x}$.

Definition 25. Let $\hat{\sigma}_{K^i,\boldsymbol{a}^i,\boldsymbol{b}^i}(\boldsymbol{x};t) := (K^i)^\top(t(T-t)\boldsymbol{a}^i(t)\odot(K^i\boldsymbol{x}+\boldsymbol{b}^i))$ for any

$\boldsymbol{x} \in \mathbb{R}^n$, $t \in [0, T]$. Any boundary-value preserving TL-SympNet $\hat{\varphi}_t$ is an alternating composition of $\mathcal{T}^{low}$, defined as in (6.22a), and a new $\hat{\mathcal{T}}^{up}$, defined as

$$\hat{\mathcal{T}}_t^{up,i} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x} + \hat{\sigma}_{K^i,\boldsymbol{a}^i,\boldsymbol{b}^i}(\boldsymbol{p}; t) \\ \boldsymbol{p} \end{pmatrix} \quad \forall \boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n \ , \tag{6.23a}$$

such that

$$\hat{\varphi}_t = \hat{\mathcal{T}}_t^{up,N} \circ \mathcal{T}_t^{low,N} \cdots \hat{\mathcal{T}}_t^{up,1} \circ \mathcal{T}_t^{low,1} \quad \text{or} \quad \hat{\varphi}_t = \mathcal{T}_t^{low,N} \circ \hat{\mathcal{T}}_t^{up,N} \cdots \mathcal{T}_t^{low,1} \circ \hat{\mathcal{T}}_t^{up,1}. \tag{6.23b}$$

Note that $\hat{\varphi}_0^{(1)}(\boldsymbol{x}, \boldsymbol{p}) = \hat{\varphi}_T^{(1)}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{x}$. Here $f^{(1)}$ denotes the first n output components of any $f : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$.

We will use boundary-value preserving TL-SympNet $\hat{\varphi}_t$ as the coordinate transformation module in TSympOCNet.

6.3.3 The TSympOCNet Method

In this subsection, we describe our algorithm of using TSympOCNet to solve the original optimal control problem together with the Hamiltonian system. We use $\hat{\varphi}_t^\theta$ instead of $\hat{\varphi}_t$ to denote the coordinate transformation (i.e., boundary-value preserving TL-SympNet), to highlight the dependence on trainable variable $\theta = (K^i, \boldsymbol{a}^i, \boldsymbol{b}^i)_{i \in \{1,\dots,N\}}$.

Let us first discretize the time horizon by $\bar{N} = \frac{T}{\Delta t}$ steps, and denote $\bar{t}_k = k\Delta t$, for $k \in \{0, 1, \dots, \bar{N}\}$. The optimal trajectory in the latent space at every $\bar{t}_k$, $(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k))$, can be computed using (6.16) and (6.17). The inverse of $(\phi_t)_{t \in [0,T]}$,

which maps the trajectory and Hamiltonian system of the latent space to the phase space, will be approximated by $(\hat{\varphi}_t^\theta)_{t\in[0,T]}$, and evaluated at data pints $\{(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k)\}$.

To learn the parameters θ, we perform iterative steps to construct and minimize a physics-informed loss function of θ. In each iteration, we randomly sample $\tilde{N}$ points among $[0, T]$, and denote the value of them by $\{t_i\}_{i\in\{1,2,\dots,\tilde{N}\}}$. Then, if $t_i \in [\bar{t}_k, \bar{t}_{k+1}[$, for some $k \in \{0, 1, \dots, \bar{N}-1\}$, we calculate the value $(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i))$ by a linear interpolation of $(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k))$ and $(\boldsymbol{y}(\bar{t}_{k+1}), \boldsymbol{q}(\bar{t}_{k+1}))$. Moreover, by the affine property of $\hat{\varphi}_t^\theta$ as we defined in (6.23), the value of $\hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i))$ can be computed by a linear interpolation of $\hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k))$ and $\hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(\bar{t}_{k+1}), \boldsymbol{q}(\bar{t}_{k+1}))$. For easy expression, we will simply denote by I^{t_i} such linear interpolation operator, i.e.,

$$\begin{aligned}(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i)) &= \hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i)) \\ &:= I^{t_i} \circ \hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k)) \ .\end{aligned} \tag{6.24}$$

Then, the derivative of $(\boldsymbol{x}_\theta(s), \boldsymbol{p}_\theta(s))$ w.r.t s at t_i follows the chain rule:

$$\begin{aligned}\begin{pmatrix}\dot{\boldsymbol{x}}_\theta(t_i) \\ \dot{\boldsymbol{p}}_\theta(t_i)\end{pmatrix} &= \mathbf{Jac}\hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i)) \begin{pmatrix}\dot{\boldsymbol{y}}(t_i) \\ \dot{\boldsymbol{q}}(t_i)\end{pmatrix} + \frac{\partial \hat{\varphi}_s^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i))}{\partial s} \mid_{t_i} \\ &= I^{t_i} \circ \mathbf{Jac}\hat{\varphi}_{t_i}^\theta(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k)) \begin{pmatrix}\dot{\boldsymbol{y}}(\bar{t}_k) \\ \dot{\boldsymbol{q}}(\bar{t}_k)\end{pmatrix} + I^{t_i} \circ \frac{\partial \hat{\varphi}_s^\theta(\boldsymbol{y}(\bar{t}_k), \boldsymbol{q}(\bar{t}_k))}{\partial s} \mid_{t_i}\end{aligned} \ , \tag{6.25}$$

where $\mathbf{Jac}\hat{\varphi}_{t_i}^\theta$ denotes the Jacobian of $\hat{\varphi}_{t_i}^\theta$ w.r.t $(\boldsymbol{y}, \boldsymbol{q})$. Notice that in implementation, $\mathbf{Jac}\hat{\varphi}_{t_i}^\theta$ and $\frac{\partial \hat{\varphi}_s^\theta}{\partial s}$ are computed via automatic differentiation, and the overall computation can be further boosted via JVP (Jacobian vector product) method in deep learning libraries, e.g., JAX. The physics-informed loss function w.r.t θ is then

defined as

$$\begin{aligned}\mathcal{L}(\theta) = &\sum_{i=0}^{N_t-1} ||\dot{\boldsymbol{x}}_\theta(t_i) - \nabla_p H_{\epsilon,l}(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i))||+ \\ &\sum_{i=0}^{N_t-1} ||\dot{\boldsymbol{p}}_\theta(t_i) + \nabla_x H_{\epsilon,l}(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i))|| \ .\end{aligned} \tag{6.26}$$

Then, in each iteration, we update θ by minimizing (6.26). The optimization is performed with stochastic gradient descent based methods, e.g., Adam [108].

It is worth mentioning that, to well handle the state constrain cases, we also perform iterative procedures to update the hyperparameters ϵ and l, defined in the penalty function (6.10) to convert to an unconstrained case. We start with fixed $\epsilon = \epsilon_0, l = l_0$. Then, we do the iterations of sampling in time horizon, computing (6.25), constructing and minimizing the loss function (6.26). We then iteratively update ϵ, l via

$$\begin{aligned}\epsilon_{j+1} &= n_1 \epsilon_j \\ l_{j+1} &= n_2 l_j\end{aligned} \tag{6.27}$$

and repeat the computation until convergence.

The overall algorithm consists of the following steps:

1. Set $s_i = \frac{iT}{N_s - 1}, i \in \{0, \cdots N_s - 1\}$. Precompute $\{(\boldsymbol{y}(s_i), \boldsymbol{q}(s_i))\}_{i=0}^{N_s-1}$ using (6.16) and (6.17).

2. Initialize hyperparameters $\epsilon = \epsilon_0, l = l_0$ in (6.10).

3. Uniformly sample $0 = t_0 \leq \cdots \leq t_{N_t-1} = T$. Calculate $\{(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i))\}_{i=1}^{N_t-1}$ by

interpolation. Compute the phase trajectory $\{(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i))\}_{i=1}^{N_t-1}$ via

$$(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i)) = \varphi_{t_i}^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i)).$$

4. Compute

$$(\boldsymbol{x}'_\theta(t_i), \boldsymbol{p}'_\theta(t_i)) = \mathbf{Jac}\varphi_{t_i}^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i)) \begin{pmatrix} \boldsymbol{y}'(t_i) \\ \boldsymbol{q}'(t_i) \end{pmatrix} + \frac{d\varphi_s^\theta(\boldsymbol{y}(t_i), \boldsymbol{q}(t_i))}{ds}(t_i),$$

where $\mathbf{Jac}\varphi_{t_i}^\theta$ and $\frac{d\varphi_t^\theta}{dt}$ are computed via automatic differentiation. $(\boldsymbol{y}'(t_i), \boldsymbol{q}'(t_i))$ are calculated using (6.15). The overall computation can be further boosted via JVP (Jacobian vector product) method in deep learning libraries, e.g., JAX.

5. Update θ by optimizing physics-informed loss

$$\begin{aligned}\mathcal{L}(\theta) = &\sum_{i=0}^{N_t-1} ||\boldsymbol{x}'_\theta(t_i) - \nabla_p H_{\epsilon,l}(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i))|| + \\ &\sum_{i=0}^{N_t-1} ||\boldsymbol{p}'_\theta(t_i) + \nabla_x H_{\epsilon,l}(\boldsymbol{x}_\theta(t_i), \boldsymbol{p}_\theta(t_i))||.\end{aligned}$$

The optimization can be done with stochastic gradient descent based methods, e.g., Adam [108].

6. Update ϵ, l via

$$\epsilon_{j+1} = n_1 \epsilon_j$$
$$l_{j+1} = n_2 l_j$$

7. Repeat step 4, 5 and 6 in total N_{iter} times till convergence.

8. Predict the rolled-out phase trajectory $(\boldsymbol{x}_\theta(s_i), \boldsymbol{p}_\theta(s_i)) = \varphi_{s_i}^\theta(\boldsymbol{y}(s_i), \boldsymbol{q}(s_i))$.

6.4 Applications in path planning problems with obstacle and collision avoidance

In this section, we apply our method to path planning problems with multiple agents. We assume that each agent is represented by a ball in the physical space with radius C_d. We set the state and control variable to be $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_M) \in \mathbb{R}^{Md_x}$, $\boldsymbol{u} = (\boldsymbol{u}_1, \dots, \boldsymbol{u}_M) \in \mathbb{R}^{Md_u}$, where M is the number of drones, and each $\boldsymbol{x}_j = (\boldsymbol{w}_j, \boldsymbol{v}_j) \in \mathbb{R}^{d_x}$ denotes the position and velocity of the center of each drone. We consider the Newtonian dynamics with quadratic resistance force $\dot{\boldsymbol{w}}_i = \boldsymbol{v}_i$, $\dot{\boldsymbol{v}}_i = \boldsymbol{u}_i - k\boldsymbol{v}_i|\boldsymbol{v}_i|$. then the dynamics can be rewritten as

$$\dot{\boldsymbol{x}}_i = \begin{pmatrix} \boldsymbol{v}_i \\ \boldsymbol{u}_i \end{pmatrix} = \begin{pmatrix} O & I \\ O & O \end{pmatrix} \boldsymbol{x}_i - k \begin{pmatrix} O & O \\ O & I \end{pmatrix} \boldsymbol{x}_i|\boldsymbol{x}_i| + \begin{pmatrix} O \\ I \end{pmatrix} \boldsymbol{u}_i. \tag{6.28}$$

To ensure the safety and smooth operation of the agents, we employ a constraint function h, comprising two components: h_1 for obstacle avoidance and h_2 for collision avoidance among agents. Let n_o denote the number of obstacles, labeled as $E_1, \dots, E_{n_o}$.

The function $h_1 \colon \mathbb{R}^{\frac{Md_x}{2}} \to \mathbb{R}^{Mn_o}$ is defined as follows:

$$h_1(\boldsymbol{w}_1, \dots, \boldsymbol{w}_M) = (D(\boldsymbol{w}_1), \dots, D(\boldsymbol{w}_M)) \quad \forall \boldsymbol{w}_1, \dots, \boldsymbol{w}_M \in \mathbb{R}^{\frac{d_x}{2}}, \tag{6.29}$$

where $D \colon \mathbb{R}^{\frac{d_x}{2}} \to \mathbb{R}^{n_o}$ is defined as:

$$D(\boldsymbol{w}) = (D_1(\boldsymbol{w}), \dots, D_{n_o}(\boldsymbol{w})) \quad \forall \boldsymbol{w} \in \mathbb{R}^{\frac{dx}{2}}, \tag{6.30}$$

and each D_j signifies the collision status with the j-th obstacle E_j; $D_j(\boldsymbol{w}) < 0$ denotes collision of the drone at position $\boldsymbol{w}$ with E_j.

For instance, D_j could be equal to the signed distance function to E_j. The precise definition of D_j relies on the shape of E_j, deferred to later sections for specific examples.

The function $h_2 \colon \mathbb{R}^{\frac{Md_x}{2}} \to \mathbb{R}^{M(M-1)/2}$ is the collision avoidance constraint function. Each component addresses collision avoidance between a pair of drones. Considering each drone as a ball centered at a point in $\mathbb{R}^{\frac{d_x}{2}}$, collision occurs when the distance between centers is less than the sum of their radii.

Thus, the l-th component of h_2 is:

$$(h_2)_l(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M) = ||\boldsymbol{w}_i - \boldsymbol{w}_j|| - (2C_d) \quad \forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_M \in \mathbb{R}^{\frac{d_x}{2}}, \tag{6.31}$$

where C_d represents the drone's radius, and $l = i + (j-1)(j-2)/2$ for any $1 \leq i < j \leq M$ denotes the corresponding constraint index for the pair (i, j).

Consequently, $(h_2)_l(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M) < 0$ indicates collision between the i-th and j-th drones, thereby facilitating collision avoidance among agents.

Training Warmup Apart from the training procedure listed in section 6.3.3, we offer an additional step which could potentially lead to faster convergence and reducing the possibility of converging to a local but not global optima. We refer to this step as *training warmup*. We empirically observed that the convergence of TSympOCNet depends on the initial condition $\boldsymbol{x}(0) = \boldsymbol{x}^{(0)} = (\boldsymbol{w}^{(0)}, \boldsymbol{v}^{(0)})$ and terminal condition $\boldsymbol{x}(T) = \boldsymbol{x}^{(T)} = (\boldsymbol{w}^{(T)}, \boldsymbol{v}^{(T)})$. Suppose that TSympOCNet converges faster on another pair of $(\tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_T)$. We can convert the optimal control problem in

(6.7) into a sequence of N_{opt} optimization problems. At step i, we solve the problem

$$\begin{aligned}
&\inf_{\boldsymbol{u}(\cdot)\in\mathcal{U}} \int_0^T \sum_{i=1}^M F_i(\boldsymbol{x}_i(s)) + G_i(\boldsymbol{u}_i(s))ds \\
&\text{s.t} \begin{cases}
\boldsymbol{x}_i(s) = \begin{pmatrix} \boldsymbol{w}_i(s) \\ \boldsymbol{v}_i(s) \end{pmatrix} \ \forall\ s \in [0,T]\ , \\
\dot{\boldsymbol{x}}_i(s) = f_i(\boldsymbol{x}_i(s)) + B_i\boldsymbol{u}_i(s),\ \forall\ s \in [0,T]\ , \\
h(\boldsymbol{w}(s)) \geq 0,\ \forall\ s \in [0,T]\ , \\
\boldsymbol{x}(0) = \dfrac{i}{N_{opt}}\boldsymbol{x}^{(0)} + \dfrac{N_{opt}-i}{N_{opt}}\tilde{\boldsymbol{x}}_0,\ \boldsymbol{x}(T) = \dfrac{i}{N_{opt}}\boldsymbol{x}^{(T)} + \dfrac{N_{opt}-i}{N_{opt}}\tilde{x}_T\ ,
\end{cases}
\end{aligned} \tag{6.32}$$

and use the obtained coordinate transformation $\varphi_s^{(i)}$ as the initialization for the next iteration. We apply this training warmup scheduler in the second example of section 6.4.1 and all examples in section 6.4.2. In our experiments, to find $(\tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{x}}_T)$, we first solve a simpler optimal control problem

$$\begin{aligned}
&\inf_{\boldsymbol{v}(\cdot)\in\mathcal{V}} \int_0^T \sum_{i=1}^M F_i(\boldsymbol{x}_i(s))ds \\
&\text{s.t} \begin{cases}
\boldsymbol{x}_i(s) = \begin{pmatrix} \boldsymbol{w}_i(s) \\ \boldsymbol{v}_i(s) \end{pmatrix} \ \forall\ s \in [0,T]\ , \\
\dot{\boldsymbol{w}}_i(s) = \boldsymbol{v}_i(s),\ \forall\ s \in [0,T]\ , \\
h(\boldsymbol{w}(s)) \geq 0,\ \forall\ s \in [0,T]\ , \\
\boldsymbol{w}(0) = \boldsymbol{w}^{(0)},\ \boldsymbol{w}(T) = \boldsymbol{w}^{(T)}
\end{cases}
\end{aligned} \tag{6.33}$$

again with TSympOCNet to get a solution $\tilde{\boldsymbol{v}}$. Then we set $\tilde{\boldsymbol{x}}_0 = (\boldsymbol{w}^{(0)}, \tilde{\boldsymbol{v}}(0))$ and $\tilde{\boldsymbol{x}}_T = (\boldsymbol{w}^{(T)}, \tilde{\boldsymbol{v}}(T))$.

Simulation results are presented in the following subsections. In section 6.4.1, we present lower-dimensional examples and benchmark the solutions with shooting

methods. In section 6.4.2, we demonstrate the capability of the existing framework to scale to 256 dimensional problems. We further compare the total running cost, constraint violation criterion and running time cost of TSympOCNet with vanilla PINN. In section 6.4.3, we consider an example with highly nonconvex obstacles.

6.4.1 Single / four agent with circular obstacle

To prevent drone collisions with obstacles, we define the constraint function h as per (6.29). The function D in (6.30) is defined by:

$$D(\boldsymbol{w}) = ||\boldsymbol{w} - \mathring{\boldsymbol{w}}|| - (C_o + C_d) \quad \forall \boldsymbol{x} \in \mathbb{R}^2,$$

where $\mathring{\boldsymbol{w}}$ and C_o represents the center and radius of the obstacle respectively. In this experiment, we set $C_o = 0.15$, $C_d = 0.05$.

Single agent Assume $M = 1$, $L(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{u}\|^2$. It is assumed that the agent starts from the top left with zero initial velocity and ends at the lower right with zero terminal velocity, i.e. $\boldsymbol{x}^{(0)} = (-0.5, 0.5, 0, 0)$ and $\boldsymbol{x}^{(T)} = (0.5, -0.5, 0, 0)$.

We compute the solutions to the optimal control problem (6.7) with resistance coefficient $k = 0, 1, 2$. In Fig. 6.3, we plot the planned trajectory by TSympOCNet $\boldsymbol{x} = (\boldsymbol{w}, \boldsymbol{v})$ together with the control signal $\boldsymbol{u}$. It can be seen that the same planned trajectory $\boldsymbol{x}$ was found with varying level of k. As resistance coefficient k increases, $\boldsymbol{u}$ exhibits prolonged alignment with the velocity $\boldsymbol{v}$, consistent with physical principles.

We further validate our training framework on this low-dimensional problem by benchmarking against the shooting method. It can be seen in Fig. 6.4 that our

solution matches the solution provided by shooting method, which means the result is a solution to the Hamiltonian ODE with H provided by (6.12).

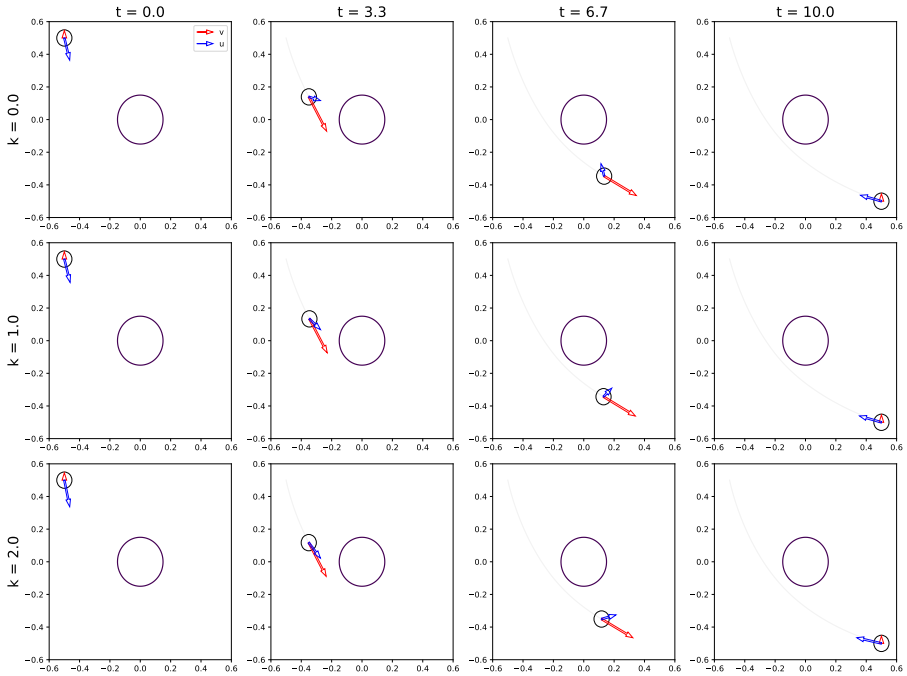


Figure 6.3: **The planned path and control at resistance coefficient** $k = 0, 1, 2$ **and time step** $t = 0, 3.3, 6.7, 10$**.** As resistance coefficient k increases, the control signal $\boldsymbol{u}$ exhibits prolonged alignment with the velocity $\boldsymbol{v}$.

Four agents Assume $M = 4$, $L(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\sum_{i=1}^{4}(\|\boldsymbol{u}_i\|^2 + \|\boldsymbol{v}_i\|^2)$, $k = 0$. The initial positions of the drones are near the boundary of the room, and the terminal positions are the opposite locations, i.e., we set $\boldsymbol{w}_T = -\boldsymbol{w}_0$. It is also assumed that the agents start and terminate with zero velocity, as in the previous example. This problem contains several asymmetric local optima and two circular symmetric global optima. We found that with the training warmup, our method converges to the global optima in all 5 independent simulations. However, if the training warmup is not applied, 3 out of 5 cases TSympOCNet converge to the local optima, as shown in Fig. 6.5. The solution is further validated against the shooting method, which exhibits great alignment, demonstrating that it is a solution to the Hamiltonian ODE, as shown in Fig. 6.6. Note that we found the shooting method doesn't con-

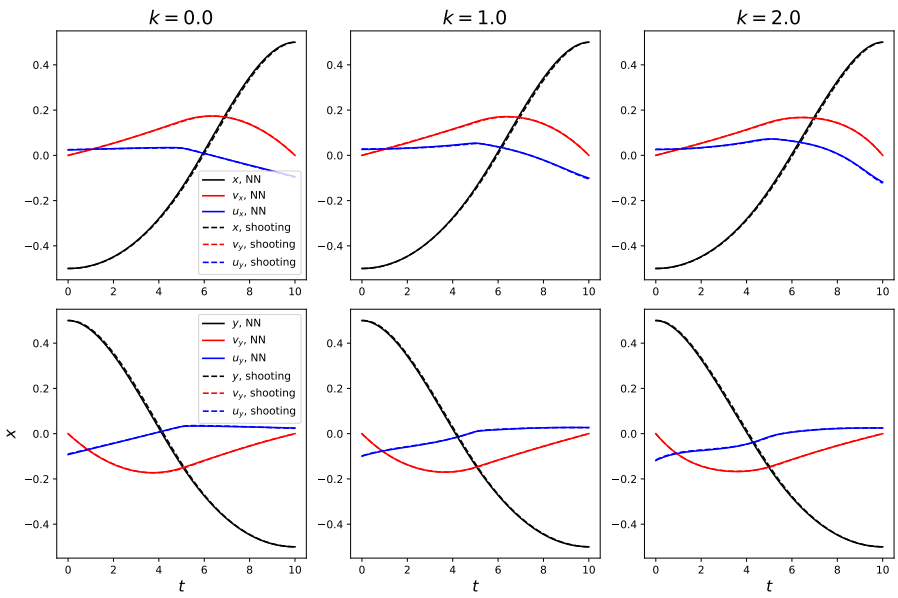


Figure 6.4: **The solution generated by TSympOCNet validated by the shooting method, in the case of 1 agent.** Our solution matches the solution provided by shooting method, which means the result is a solution to the Hamiltonian ODE with H provided by (6.12).

verge well for the $2Md_x = 32$ dimensional Hamiltonian ODE. So we first use the shooting method to find the solution for one agent without considering the mutual collision avoidance requirement characterized by h_2. Then we rotate that solution by $90°, 180°, 270°$ respectively to obtain the solution for all 4 agents.

6.4.2 High dimensional problem with Newtonian dynamics

We consider the path planning problem of agents in a room of size $[-0.5, 0.5] \times [-0.5, 0.5]$. Collisions among the agents and collisions between agents and the room walls are prevented. The function D in (6.30) is defined by:

$$D_{bd}(\boldsymbol{w}) = (w_1 + 0.5, 0.5 - w_1, w_2 + 0.5, 0.5 - w_2) \quad \forall \boldsymbol{w} = (w_1, w_2) \in \mathbb{R}^2.$$

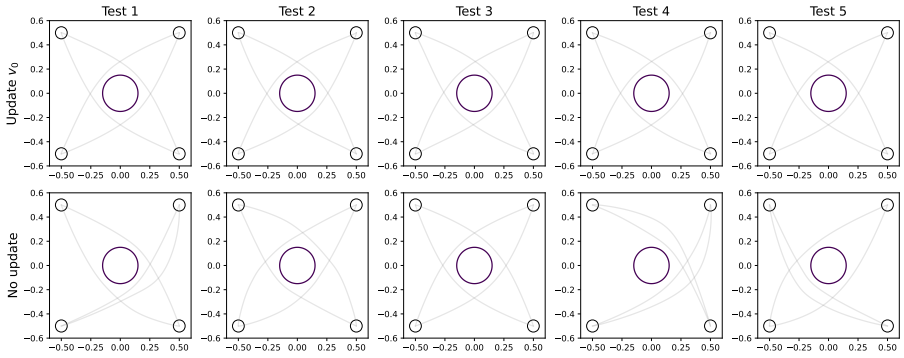


Figure 6.5: **Comparison of planned trajectory with or without the warmup, at 5 different initializations.** The figure shows that our algorithm is more robust to random initialization and converges more often to global minima when the warmup scheduler is applied.

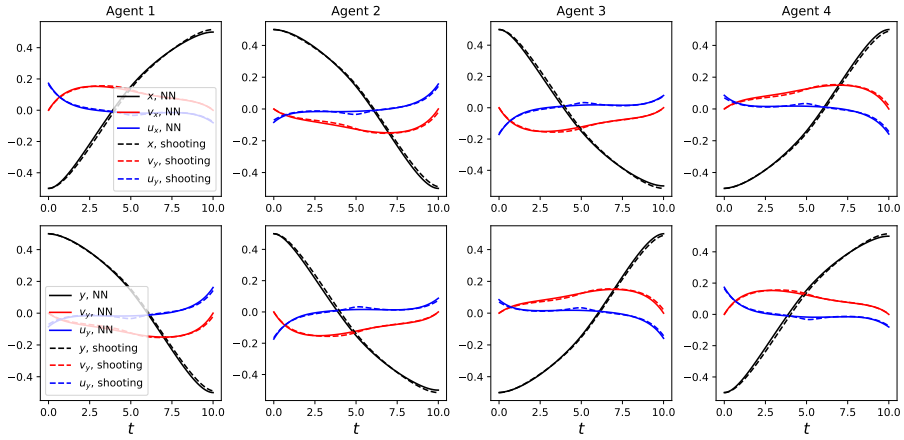


Figure 6.6: **The solution generated by our algorithm validated by the shooting method, in the case of 4 agents.** Our solution matches the solution provided by shooting method, which means the result is a solution to the Hamiltonian ODE.

In this experiment, we set $C_d = 0.02$. We assume $L(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{u}\|^2$, $k = 0$. The drones' initial positions lie near the room boundary, and their terminal positions are opposite, denoted as $\boldsymbol{w}_T = -\boldsymbol{w}_0$. This constitutes a high-dimensional example (state space dimension $n = 4M$, varying from 32 to 256 in our experiments). We run 10 repeated experiments with number of agents $M = 8, 16, 32, 64$ to test the robustness of TSympOCNet. We further compare the solution obtained from TSympOCNet with solution obtained from PINN. For both neural network architectures, we use the same training algorithm in 6.3.3, both with training warmup scheduler. The

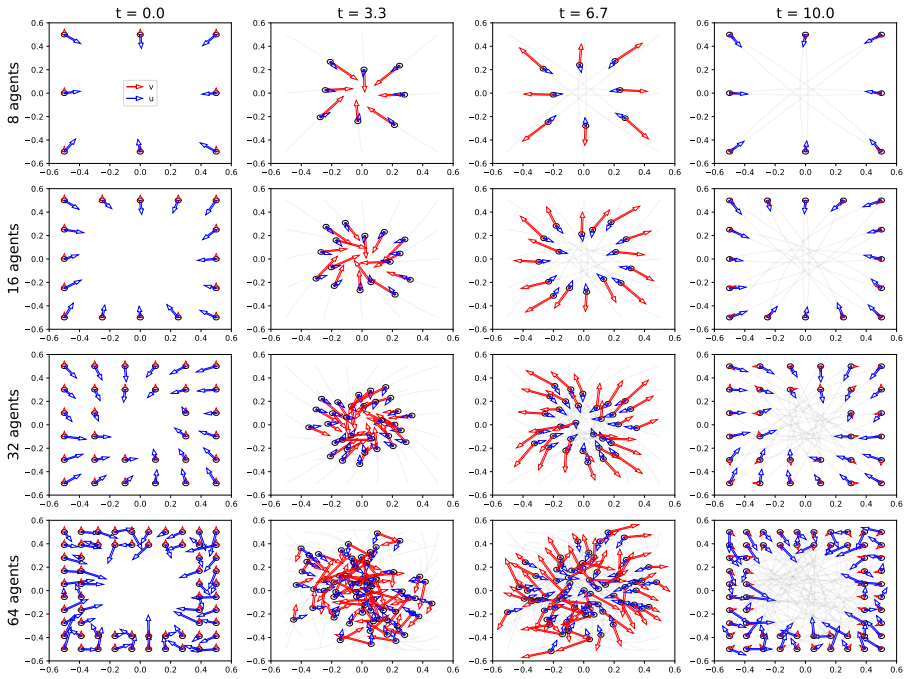


Figure 6.7: **The planned path and control for $M = 8, 16, 32, 64$ agents and time step $t = 0, 3.3, 6.7, 10$.**

only difference between two approaches lies in the parameterization of $(\boldsymbol{x}_\theta(s), \boldsymbol{p}_\theta(s))$. TSympOCNet represents it by composition of latent LQR solution and TL-SympNet, while PINN sets $(\boldsymbol{x}_\theta(s), \boldsymbol{p}_\theta(s))$ as a fully-connected neural network on s. In each trial, we document the training time in seconds, the running cost $L(\boldsymbol{x}, \boldsymbol{u})$ and the constraint violation metric

$$\mathcal{D} = \min_s \min_{1 \le i < j \le M} ||\boldsymbol{w}_i(s) - \boldsymbol{w}_j(s)|| - 2C_d, \tag{6.34}$$

where $\boldsymbol{w}_i(s)$ denotes the center position for the i-th drone at time s. The mean and standard deviation of these three metrics are reported in Table 6.1. It can be seen that TSympOCNet consistently outperform vanilla PINN in terms of generating a trajectory with lower running cost and lower constraint violation. It is worth remarking that in the case of $M = 8, 16$ agents, the constraint is exactly satisfied, which means no collision is going to happen. However, in the case of 64 agents, the

constraints is slightly violated in most of the solutions provided by TSympOCNet. To provide a feasible solution, one need to rescale the problem and set $C_d = 0.16$. On the other hand, collision always occurs in PINN solutions when $M = 64$. The solution trajectory by TSympOCNet in one trial is shown in Fig. 6.7.

We dive deeper into the solutions provided by PINN and TSympOCNet, focusing on cases where $M = 4$ and $M = 64$, as illustrated in Fig. 6.8. When $M = 4$, TSympOCNet's planned path exhibits a distinct behavior: the agents initially converge towards the origin, then just before encountering one another, initiating a synchronized rotation before dispersing. Conversely, PINN's path lacks this rotational symmetry—One pair of agents accelerates, passing ahead, while the second pair decelerates, awaiting the first pair's passage before proceeding. The first solution is more optimal in terms of running cost. In scenarios where $M = 64$, TSympOCNet's planned path demonstrates fewer constraint violations compared to PINN.

# agents		8	16	32	64
PINN	Runtime (s)	338 ± 2	338 ± 2	354 ± 2	368 ± 2
	$L(\boldsymbol{x}, \boldsymbol{u})$	0.080 ± 0.007	0.175 ± 0.013	0.346 ± 0.020	4.516 ± 1.091
	$\mathcal{D}$	$(-1 \pm 2) \times 10^{-4}$	$(-4 \pm 6) \times 10^{-4}$	$(-2 \pm 3) \times 10^{-3}$	$(-39 \pm 1) \times 10^{-3}$
TSympOCNet	Runtime (s)	465 ± 2	504 ± 2	588 ± 67	529 ± 51
	$L(\boldsymbol{x}, \boldsymbol{u})$	0.072 ± 0.0003	0.152 ± 0.006	0.304 ± 0.049	1.125 ± 0.118
	$\mathcal{D}$	0	0	$(-2 \pm 4) \times 10^{-4}$	$(-8 \pm 1) \times 10^{-3}$

Table 6.1: **Comparison of vannila PINN and TSympOCNet on problems of dimension** $8, 16, 32, 64$**.** Under the provided hyperparameter setup, the solution obtained by TSympOCNet is closer to the global optima compared to vannila PINN. The collision-avoidance constraint is also satisfied better.

6.4.3 Planning in nonconvex environment

In this experiment, we set $C_d = 0.02$. We assume $L(\boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{u}\|^2$, $k = 0$. We consider the path planning problem of agents in a room of size $[-0.5, 0.5] \times [-0.5, 0.5]$. Apart from collisions among the agents and collisions between agent and the room boundary described in the previous section, we'd also like to prevent the collision

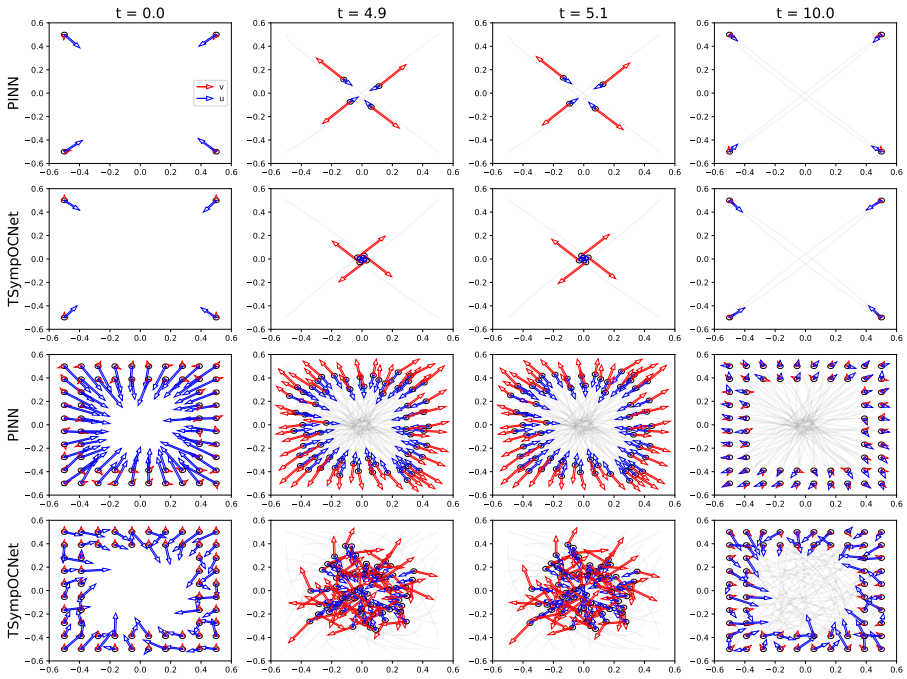


Figure 6.8: **The planned path and control for $M = 4$ and 64 agents using PINN and TSympOCNet.** TSympOCNet produces a solution with lower cost in lower-dimensional problems and more feasible solution in higher dimensions.

between agents and several walls inside the room, which created a maze. The shape of the maze can be seen in Fig. 6.9.

$$D(\boldsymbol{w}) = (D_{bd}(\boldsymbol{w}), D_{maze}(\boldsymbol{w}))$$

$$D_{maze}^{(j)}(\boldsymbol{w}) = \min_{\boldsymbol{y}\in l_j} \|\boldsymbol{w}-\boldsymbol{y}\| - (C_o + C_d) \quad \forall \boldsymbol{w} \in \mathbb{R}^2.$$

The expression $\min_{\boldsymbol{y}\in l_j} \|\boldsymbol{w}-\boldsymbol{y}\|$ in D_{maze} calculates the distance between point $\boldsymbol{w}$ and the line segment l_j. Considering that obstacle E_j encompasses all points within a distance of C_o from line segment l_j, a drone collides with E_j if and only if the distance between its center and l_j is less than $C_o + C_d$. Thus, the function $D_{maze}^{(j)}$ defined in this manner imposes a constraint that prevents drone collisions with the j-th obstacle E_j.

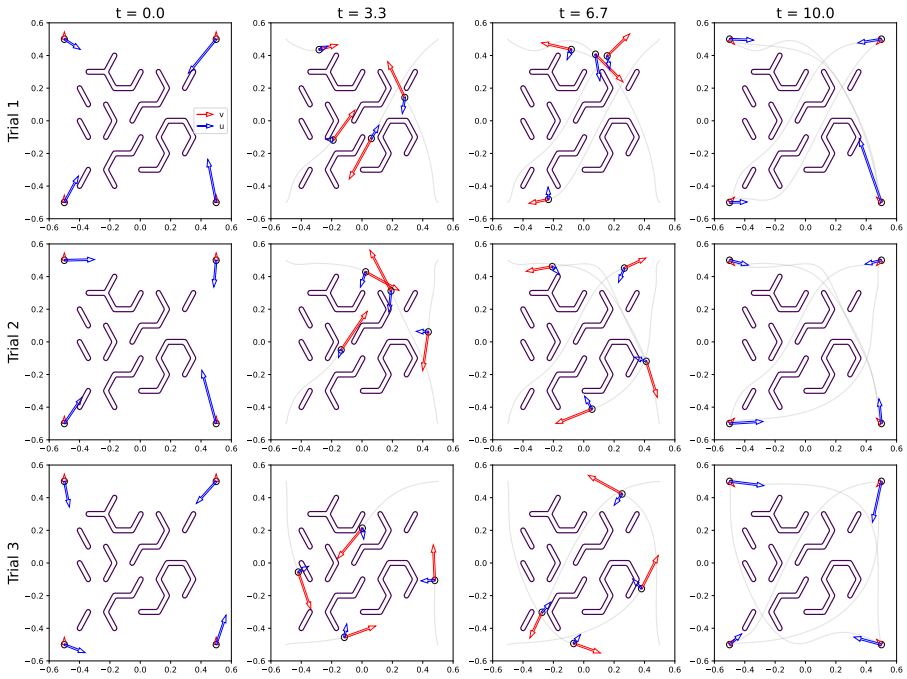


Figure 6.9: **The planned path and control in the nonconvex maze.** We run three independent simulations and obtained three different solutions. The running cost $L(\boldsymbol{x}, \boldsymbol{u}) = 0.104, 0.151, 0.193$ in trial $1, 2, 3$. In all cases, no constraint violations are observed. TSympOCNet may converge to suboptimal solutions in this numerical example.

We ran three independent simulations and observed three different solutions, as shown in Fig. 6.9. The running cost $L(\boldsymbol{x}, \boldsymbol{u}) = 0.104, 0.151, 0.193$ in trial $1, 2, 3$. In all cases, no constraint violations are observed. It is worth noting that TSympOCNet may converge to suboptimal solutions in this numerical example.

6.5 Summary

We've introduced TSympOCnet, an extension of SympOCNet for tackling high-dimensional optimal control problems with state constraints and general dynamics. Applying TSympOCnet to multi-agent simultaneous path planning tasks with obstacle avoidance demonstrates its efficacy in solving high-dimensional problems in

256 dimensions. These findings highlight TSympOCnet's potential for real-time solutions in high-dimensional optimal control problems. In future research, we aim to explore extending the current framework to handle feedback control problems by solving the latent LQR problem via Riccati solvers.

Chapter Seven

Summary

This thesis explores data-driven discovery of dynamical systems' governing equations using neural networks, following a physics-informed approach. It focuses on embedding physical principles directly into neural network architectures to enhance model stability and generalization. The work introduces three specialized networks: symplectic networks (SympNets) for preserving symplecticity in phase space, Poisson neural networks (PNNs) for maintaining the Poisson structure, and GENERIC formalism informed neural networks (GFINNs) for combining conservative and dissipative dynamics, alongside Symplectic Graph Neural Networks (SympGNNs) for preserving both symplecticity and permutation equivariance.

A notable application of SympNets demonstrated is in optimal control problems, leveraging symplectic transformations to simplify Hamiltonian ODEs for efficient path planning, including a significant instance of managing 256 drones. The thesis progresses from discussing the theoretical basis and application of these networks to showcasing their capability in modeling differential equations and solving complex control problems, highlighting their potential in bridging deep learning and the study of dynamical systems as well as optimal control.

There are several possible future directions following the research presented in this thesis. Some important topics that are worth exploring include the system identification, distribution sampling and optimal control problems in *high dimensions*, where traditional numerical schemes and purely data-driven approaches without inductive biases inevitably face the so called curse-of-dimensionality.

The general question we'd like to pose is: How to identify a map $f : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$, using a limited $k \ll (nd)^2$ number of data, when $n, d \gg 1$? Here f, n and d could have different interpretations based on the context:

- **Data-driven discovery of dynamics**: Here, the goal is to determine the operator f such that $f(x(t)) = x(t+h)$, enabling f to act as a proxy model that lessens the computational demands of particle simulations. In this case, n represents the count of particles, while d denotes the physical dimensions of an individual particle.

- **UAV Path planning**: In this scenario, f represents the transformation from the phase trajectory to more straightforward, disentangled trajectories within a latent space. Here, n refers to the number of UAVs or drones, and d signifies the dimension of the phase trajectory for an individual UAV.

- **Distribution sampling**: For this application, f is understood as a normalizing flow that translates complex distributions into simpler forms, such as i.i.d. Gaussian distributions. The term n stands for the number of random variables, and d represents the intrinsic dimension of each variable.

The question above is obviously underdetermined so additional constraints are required. We further pose three followup questions:

- How do we decide on the appropriate constraints or inductive biases to integrate into the model f_θ that seeks to mimic f?

- Upon recognizing that certain inductive biases are indispensable, is it feasible to construct a parameterized family of functions that not only adheres to these constraints but also retains the capacity to universally approximate?

- Is it possible to ensure that the models f_θ we develop are interpretable, providing clear insights into what are the governing laws of physics?

Regarding the first question, our research has entertained the idea of incorporating concepts such as symplecticity, Poisson structures, GENERIC formalisms, and permutation equivariance into neural network designs. However, the applicability of these formalisms in practical scenarios often remains a mystery. It's imperative to determine if a quantitative metric exists that can assess to what extent these inductive biases align with the dataset. Such a metric would enable us to pre-select the most suitable formalism before deploying the corresponding model.

Regarding the second question, one future direction we are going to investigate is whether one can approximate a symplectically permutation equivariant function with guaranteed universal approximation theorem. In Chapter 2, we demonstrate that SympGNNs are symplectically permutation equivariant, but there is no guarantee of its expressive power. Inspired by theorem 6, we propose the following conjecture:

Conjecture 1. Suppose $V_i : \mathbb{R}^{n\times d} \to \mathbb{R}$ and $T_i : \mathbb{R}^{n\times d} \to \mathbb{R}$ are permutation invariant *and can approximate arbitrary permutation invariant functions* for all $i = 1, \cdots, l$, then the alternating composition of the following two parameterized functions:

$$\begin{aligned}
&\mathcal{E}_i^{low}\begin{pmatrix}\boldsymbol{p} & \boldsymbol{q}\end{pmatrix} = \begin{pmatrix}\boldsymbol{p} & \boldsymbol{q} + \nabla T_i(\boldsymbol{p})\end{pmatrix} \quad \forall \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{n\times d},\\
&\mathcal{E}_i^{up}\begin{pmatrix}\boldsymbol{p} & \boldsymbol{q}\end{pmatrix} = \begin{pmatrix}\boldsymbol{p} + \nabla V_i(\boldsymbol{q}) & \boldsymbol{q}\end{pmatrix} \quad \forall \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{n\times d}\\
&\varphi = \mathcal{E}_l^{up} \circ \mathcal{E}_l^{low} \cdots \mathcal{E}_1^{up} \circ \mathcal{E}_1^{low} \quad \text{or} \quad \varphi = \mathcal{E}_l^{low} \circ \mathcal{E}_l^{up} \cdots \mathcal{E}_1^{low} \circ \mathcal{E}_1^{up},
\end{aligned} \tag{7.1}$$

can approximate arbitrary symplectically permutation equivariant functions.

Should this conjecture be validated, it would pave the way for developing neural network architectures capable of approximating any symplectically permutation equivariant function. By employing deep sets [193] (or energy transformers [89]) as T_i and V_i, which are known to approximate arbitrary symplectically permutation invari-

ant functions, we can address various high-dimensional system identification problem and optimal control problems discussed in Chapters 5 and 6. Importantly, the permutation equivariance of coordinate transformations, reflecting identical dynamics across agents, would naturally extend our SympOCNet/TSympOCNet framework for feedback control solutions, not just open-loop scenarios. In the current framework, learning a transformation φ that facilitates planning with multiple initial conditions and enables feedback control requires sampling initial conditions from $U \in \mathbb{R}^{Md_x}$, where U is an open set of a high-dimensional space, which is computationally challenging. However, by incorporating a constraint on the transformation to sample from a product space $\prod_{i=1}^{M} U_i$, $U_i \in \mathbb{R}^{d_x}$, which is computationally more feasible, this approach holds promise for more efficient solutions to feedback control problems. Apart from path planning applications, such models can also be used to sample exchangable distributions via a normalizing flow framework.

In addressing the third query, we propose two potential avenues for future exploration. Firstly, employing symbolic regression models as a tool for knowledge distillation presents a promising strategy. By applying these models, we aim to derive explicit formulas from our otherwise opaque network models. This approach holds the potential to unveil an interpretable function f, thus bridging the gap between complex models and understandable, actionable insights. Secondly, an intriguing line of investigation involves examining why SympGNNs are particularly adept at overcoming oversmoothing issues and handling heterophilic data in graph-based models. A key to understanding their success lies in examining the Hamiltonian structure that underpins their functionality. A detailed study of how this Hamiltonian framework interacts with and impacts various components—such as the Dirichlet energy, which is indicative of feature smoothness—might reveal the fundamental reasons behind their effectiveness.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., *Tensorflow: A system for large-scale machine learning*, in 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 2016, pp. 265–283.

[2] M. Akian, R. Bapat, and S. Gaubert, *Max-plus algebra*, Handbook of linear algebra, 39 (2006).

[3] M. Akian, S. Gaubert, and A. Lakhoua, *The max-plus finite element method for solving deterministic optimal control problems: basic properties and convergence analysis*, SIAM Journal on Control and Optimization, 47 (2008), pp. 817–848.

[4] A. Alla, M. Falcone, and L. Saluzzi, *An efficient DP algorithm on a tree-structure for finite horizon optimal control problems*, SIAM Journal on Scientific Computing, 41 (2019), pp. A2384–A2406.

[5] A. Alla, M. Falcone, and S. Volkwein, *Error analysis for POD approximations of infinite horizon problems via the dynamic programming approach*, SIAM Journal on Control and Optimization, 55 (2017), pp. 3091–3115.

[6] J. Anderson, I. Kevrekidis, and R. Rico-Martinez, *A comparison of recurrent training algorithms for time series analysis and system identification*, Computers & chemical engineering, 20 (1996), pp. S751–S756.

[7] M. A. Armstrong, *Basic topology*, Springer Science & Business Media, 2013.

[8] A. Bachouch, C. Huré, N. Langrené, and H. Pham, *Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications*, arXiv preprint arXiv:1812.05916, (2018).

[9] S. Bansal and C. J. Tomlin, *Deepreach: A deep learning approach to high-dimensional reachability*, in 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 1817–1824.

[10] M. Bardi and I. Capuzzo-Dolcetta, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, Systems & Control: Foundations & Applications, Birkhäuser Boston, Inc., Boston, MA, 1997. With appendices by Maurizio Falcone and Pierpaolo Soravia.

[11] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, *Invertible residual networks*, in Proceedings of the 36th International Conference on Machine Learning, vol. 97 of Proceedings of Machine Learning Research, Long Beach, California, USA, 09–15 Jun 2019, PMLR, pp. 573–582.

[12] R. E. Bellman, *Adaptive control processes: a guided tour*, Princeton university press, 1961.

[13] T. Bertalan, F. Dietrich, I. Mezić, and I. G. Kevrekidis, *On learning Hamiltonian systems from data*, Chaos: An Interdisciplinary Journal of Nonlinear Science, 29 (2019), p. 121107.

[14] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Elsevier, 1982.

[15] D. P. Bertsekas, *Reinforcement learning and optimal control*, Athena Scientific, Belmont, Massachusetts, (2019).

[16] O. Bokanowski, N. Gammoudi, and H. Zidani, *Optimistic Planning Algorithms For State-Constrained Optimal Control Problems*. working paper or preprint, July 2021.

[17] O. Bokanowski, J. Garcke, M. Griebel, and I. Klompmaker, *An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations*, Journal of Scientific Computing, 55 (2013), pp. 575–605.

[18] R. Bondesan and A. Lamacraft, *Learning symmetries of classical integrable systems*, arXiv preprint arXiv:1906.04645, (2019).

[19] J. Bongard and H. Lipson, *Automated reverse engineering of nonlinear dynamical systems*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 9943–9948.

[20] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proceedings of the national academy of sciences, 113 (2016), pp. 3932–3937.

[21] B. Chamberlain, J. Rowbottom, M. I. Gorinova, M. Bronstein, S. Webb, and E. Rossi, *Grand: Graph neural diffusion*, in International Conference on Machine Learning, PMLR, 2021, pp. 1407–1418.

[22] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, *Reversible architectures for arbitrarily deep residual neural networks*, arXiv preprint arXiv:1709.03698, (2017).

[23] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, *Reversible architectures for arbitrarily deep residual neural networks*, in Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, AAAI Press, 2018, pp. 2811–2818.

[24] M. Chen, J. F. Fisac, S. Sastry, and C. J. Tomlin, *Safe sequential path planning of multi-vehicle systems via double-obstacle Hamilton-Jacobi-Isaacs variational inequality*, in 2015 European Control Conference (ECC), IEEE, 2015, pp. 3304–3309.

[25] M. Chen, Q. Hu, J. F. Fisac, K. Akametalu, C. Mackin, and C. J. Tomlin, *Reachability-based safety and goal satisfaction of unmanned aerial platoons on air highways*, Journal of Guidance, Control, and Dynamics, 40 (2017), pp. 1360–1373.

[26] M. Chen and C. J. Tomlin, *Exact and efficient Hamilton-Jacobi reachability for decoupled systems*, in 2015 54th IEEE Conference on Decision and Control (CDC), IEEE, 2015, pp. 1297–1303.

[27] P. Chen, J. Darbon, and T. Meng, *Hopf-type representation formulas and efficient algorithms for certain high-dimensional optimal control problems*, arXiv preprint arXiv:2110.02541, (2021).

[28] ——, *Lax-Oleinik-type formulas and efficient algorithms for certain high-dimensional optimal control problems*, arXiv preprint arXiv:2109.14849, (2021).

[29] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, in Advances in neural information processing systems, 2018, pp. 6571–6583.

[30] T. Chen and H. Chen, *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*, IEEE Transactions on Neural Networks, 6 (1995), pp. 911–917.

[31] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, in NeurIPS, 2018, pp. 6572–6583.

[32] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou, *Symplectic recurrent neural networks*, in 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.

[33] A. R. Conn, N. I. Gould, and P. Toint, *A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds*, SIAM Journal on Numerical Analysis, 28 (1991), pp. 545–572.

[34] M. Coupechoux, J. Darbon, J.-M. Kélif, and M. Sigelle, *Optimal trajectories of a UAV base station using Lagrangian mechanics*, in IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2019, pp. 626–631.

[35] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, *Lagrangian neural networks*, arXiv preprint arXiv:2003.04630, (2020).

[36] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems, 2 (1989), pp. 303–314.

[37] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Math. Control Signal, 2 (1989), pp. 303–314.

[38] J. Darbon, *On convex finite-dimensional variational methods in imaging sciences and Hamilton–Jacobi equations*, SIAM Journal on Imaging Sciences, 8 (2015), pp. 2268–2293.

[39] J. Darbon, P. M. Dower, and T. Meng, *Neural network architectures using min plus algebra for solving certain high dimensional optimal control problems and Hamilton-Jacobi PDEs*, arXiv preprint arXiv:2105.03336, (2021).

[40] J. Darbon, G. P. Langlois, and T. Meng, *Overcoming the curse of dimensionality for some Hamilton-Jacobi partial differential equations via neural network architectures*, Res. Math. Sci., 7 (2020), p. 20.

[41] J. Darbon and T. Meng, *On decomposition models in imaging sciences and multi-time Hamilton–Jacobi partial differential equations*, SIAM Journal on Imaging Sciences, 13 (2020), pp. 971–1014.

[42] J. Darbon and T. Meng, *On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations*, Journal of Computational Physics, 425 (2021), p. 109907.

[43] J. Darbon, T. Meng, and E. Resmerita, *On Hamilton-Jacobi PDEs and image denoising models with certain non-additive noise*, arXiv preprint arXiv:2105.13997, (2021).

[44] J. Darbon and S. Osher, *Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere*, Res Math Sci Research in the Mathematical Sciences, 3 (2016), pp. 1–26.

[45] M. A. De Gosson, *Symplectic geometry and quantum mechanics*, vol. 166, Springer Science & Business Media, 2006.

[46] D. Delahaye, S. Puechmorel, P. Tsiotras, and E. Féron, *Mathematical models for aircraft trajectory design: A survey*, in Air Traffic Management and Systems, Springer, 2014, pp. 205–247.

[47] J. Denk and G. Schmidt, *Synthesis of a walking primitive database for a humanoid robot using optimal control techniques*, in Proceedings of IEEE-RAS International Conference on Humanoid Robots, 2001, pp. 319–326.

[48] I. Deshpande, Z. Zhang, and A. G. Schwing, *Generative modeling using the sliced wasserstein distance*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 3483–3491.

[49] F. Di Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein, *Understanding convolution on graphs via energies*, arXiv preprint arXiv:2206.10991, (2022).

[50] F. Dietrich, A. Makeev, G. Kevrekidis, N. Evangelou, T. Bertalan, S. Reich, and I. G. Kevrekidis, *Learning effective stochastic differential equations from microscopic simulations: combining stochastic numerics and deep learning*, arXiv preprint arXiv:2106.09004, (2021).

[51] L. Dinh, D. Krueger, and Y. Bengio, *NICE: non-linear independent components estimation*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings, 2015.

[52] L. Dinh, J. Sohl-Dickstein, and S. Bengio, *Density estimation using real NVP*, in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.

[53] D. DiPietro, S. Xiong, and B. Zhu, *Sparse symplectically integrated neural networks*, Advances in Neural Information Processing Systems, 33 (2020).

[54] B. Djeridane and J. Lygeros, *Neural approximation of PDE solutions: An application to reachability computations*, in Proceedings of the 45th IEEE Conference on Decision and Control, Dec 2006, pp. 3034–3039.

[55] S. Dolgov, D. Kalise, and K. K. Kunisch, *Tensor decomposition methods for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM Journal on Scientific Computing, 43 (2021), pp. A1625–A1650.

[56] P. M. Dower, W. M. McEneaney, and H. Zhang, *Max-plus fundamental solution semigroups for optimal control problems*, in 2015 Proceedings of the Conference on Control and its Applications, SIAM, 2015, pp. 368–375.

[57] W. E, *A proposal on machine learning via dynamical systems*, Communications in Mathematics and Statistics, 5 (2017), pp. 1–11.

[58] A. El Khoury, F. Lamiraux, and M. Taix, *Optimal motion planning for humanoid robots*, in 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 3136–3141.

[59] P. Espanol and M. Revenga, *Smoothed dissipative particle dynamics*, Physical Review E, 67 (2003), p. 026705.

[60] L. C. Evans, *An introduction to mathematical optimal control theory version 0.2*, Lecture notes available at http://math. berkeley. edu/˜ evans/control. course. pdf, (1983).

[61] F. Fahroo and I. M. Ross, *Costate estimation by a Legendre pseudospectral method*, Journal of Guidance, Control, and Dynamics, 24 (2001), pp. 270–277.

[62] ——, *Direct trajectory optimization by a Chebyshev pseudospectral method*, Journal of Guidance, Control, and Dynamics, 25 (2002), pp. 160–166.

[63] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, T. Koolen, P. Marion, M. Posa, A. Valenzuela, K.-T. Yu, J. Shah,

K. Iagnemma, R. Tedrake, and S. Teller, *An architecture for online affordance-based perception and whole-body planning*, Journal of Field Robotics, 32 (2015), pp. 229–254.

[64] E. Faou, V. Gradinaru, and C. Lubich, *Computing semiclassical quantum dynamics with hagedorn wavepackets*, SIAM Journal on Scientific Computing, 31 (2009), pp. 3027–3041.

[65] K. Feng, *On difference schemes and symplectic geometry*, in Proceedings of the 5th international symposium on differential geometry and differential equations, 1984.

[66] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, *Optimization based full body control for the atlas robot*, in 2014 IEEE-RAS International Conference on Humanoid Robots, IEEE, 2014, pp. 120–127.

[67] M. Finzi, K. A. Wang, and A. G. Wilson, *Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints*, Advances in Neural Information Processing Systems, 33 (2020).

[68] S. Fiori, *Lie-group-type neural system learning by manifold retractions*, Neural Networks, 21 (2008), pp. 1524–1529.

[69] ——, *Extended Hamiltonian learning on Riemannian manifolds: Numerical aspects*, IEEE Transactions on Neural Networks and Learning Systems, 23 (2011), pp. 7–21.

[70] ——, *Extended Hamiltonian learning on Riemannian manifolds: Theoretical aspects*, IEEE transactions on neural networks, 22 (2011), pp. 687–700.

[71] ——, *A Riemannian steepest descent approach over the inhomogeneous symplectic group: Application to the averaging of linear optical systems*, Applied Mathematics and Computation, 283 (2016), pp. 251–264.

[72] S. Fiori and S. Prifti, *Exact low-order polynomial expressions to compute the Kolmogoroff–Nagumo mean in the affine symplectic group of optical transference matrices*, Linear and Multilinear Algebra, 65 (2017), pp. 840–856.

[73] W. Fleming and W. McEneaney, *A max-plus-based algorithm for a Hamilton–Jacobi–Bellman equation of nonlinear filtering*, SIAM Journal on Control and Optimization, 38 (2000), pp. 683–710.

[74] K. Fujiwara, S. Kajita, K. Harada, K. Kaneko, M. Morisawa, F. Kanehiro, S. Nakaoka, and H. Hirukawa, *An optimal planning of falling motions of a humanoid robot*, in 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2007, pp. 456–462.

[75] J. Garcke and A. Kröner, *Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids*, Journal of Scientific Computing, 70 (2017), pp. 1–28.

[76] D. Garg, *Advances in global pseudospectral methods for optimal control*, PhD thesis, University of Florida Gainesville, FL, 2011.

[77] S. Gaubert, W. McEneaney, and Z. Qu, *Curse of dimensionality reduction in max-plus based approximation methods: Theoretical estimates and improved pruning algorithms*, in 2011 50th IEEE Conference on Decision and Control and European Control Conference, IEEE, 2011, pp. 1054–1061.

[78] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Message passing neural networks*, Machine learning meets quantum physics, (2020), pp. 199–214.

[79] G. H. Golub and V. Pereyra, *The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate*, SIAM Journal on numerical analysis, 10 (1973), pp. 413–432.

[80] R. González-García, R. Rico-Martínez, and I. Kevrekidis, *Identification of distributed parameter systems: A neural net based approach*, Computers & Chemical Engineering, 22 (1998), pp. S965 – S968. European Symposium on Computer Aided Process Engineering-8.

[81] S. Greydanus, M. Dzamba, and J. Yosinski, *Hamiltonian neural networks*, in Advances in Neural Information Processing Systems, 2019, pp. 15353–15363.

[82] M. Grmela and H. C. Öttinger, *Dynamics and thermodynamics of complex fluids. i. development of a general formalism*, Physical Review E, 56 (1997), p. 6620.

[83] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2017), p. 014004.

[84] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, vol. 31, Springer Science & Business Media, 2006.

[85] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proceedings of the National Academy of Sciences, 115 (2018), pp. 8505–8510.

[86] Q. Hernandez, A. Badias, D. Gonzalez, F. Chinesta, and E. Cueto, *Deep learning of thermodynamics-aware reduced-order models from data*, Computer Methods in Applied Mechanics and Engineering, 379 (2021), p. 113763.

[87] Q. Hernández, A. Badías, D. González, F. Chinesta, and E. Cueto, *Structure-preserving neural networks*, Journal of Computational Physics, 426 (2021), p. 109950.

[88] M. Hofer, M. Muehlebach, and R. D'Andrea, *Application of an approximate model predictive control scheme on an unmanned aerial vehicle*, in 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2016, pp. 2952–2957.

[89] B. Hoover, Y. Liang, B. Pham, R. Panda, H. Strobelt, D. H. Chau, M. Zaki, and D. Krotov, *Energy transformer*, Advances in Neural Information Processing Systems, 36 (2024).

[90] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural networks, 2 (1989), pp. 359–366.

[91] K. Hornik, M. Stinchcombe, and H. White, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural Networks, 3 (1990), pp. 551 – 560.

[92] M. B. Horowitz, A. Damle, and J. W. Burdick, *Linear Hamilton Jacobi Bellman equations in high dimensions*, in 53rd IEEE Conference on Decision and Control, IEEE, 2014, pp. 5880–5887.

[93] C. Huré, H. Pham, and X. Warin, *Some machine learning schemes for high-dimensional nonlinear PDEs*, arXiv preprint arXiv:1902.01599, (2019).

[94] C. Huré, H. Pham, A. Bachouch, and N. Langrené, *Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis*, SIAM Journal on Numerical Analysis, 59 (2021), pp. 525–557.

[95] F. Jiang, G. Chou, M. Chen, and C. J. Tomlin, *Using neural networks to compute approximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions*, arXiv preprint arXiv:1611.03158, (2016).

[96] L. Jin, S. Li, J. Yu, and J. He, *Robot manipulator control using neural networks: A survey*, Neurocomputing, 285 (2018), pp. 23 – 34.

[97] P. Jin, Z. Lin, and B. Xiao, *Optimal unit triangular factorization of symplectic matrices*, Linear Algebra and its Applications, 650 (2022), pp. 236–247.

[98] P. Jin, L. Lu, Y. Tang, and G. E. Karniadakis, *Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness*, arXiv preprint arXiv:1905.11427, (2019).

[99] P. Jin, Y. Tang, and A. Zhu, *Unit triangular factorization of the matrix symplectic group*, arXiv preprint arXiv:1912.10926, (2019).

[100] ——, *Unit triangular factorization of the matrix symplectic group*, SIAM Journal on Matrix Analysis and Applications, 41 (2020), pp. 1630–1650.

[101] P. Jin, Z. Zhang, I. G. Kevrekidis, and G. E. Karniadakis, *Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks*, arXiv preprint arXiv:2012.03133, (2020).

[102] P. Jin, Z. Zhang, A. Zhu, Y. Tang, and G. E. Karniadakis, *SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems*, Neural Networks, 132 (2020), pp. 166–179.

[103] D. Kalise, S. Kundu, and K. Kunisch, *Robust feedback control of nonlinear pdes by numerical approximation of high-dimensional Hamilton–Jacobi–Isaacs equations*, SIAM Journal on Applied Dynamical Systems, 19 (2020), pp. 1496–1524.

[104] D. Kalise and K. Kunisch, *Polynomial approximation of high-dimensional Hamilton–Jacobi–Bellman equations and applications to feedback control of semilinear parabolic PDEs*, SIAM Journal on Scientific Computing, 40 (2018), pp. A629–A652.

[105] W. Kang and L. C. Wilcox, *Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and HJB equations*, Computational Optimization and Applications, 68 (2017), pp. 289–315.

[106] P. Kidger, J. Morrill, J. Foster, and T. J. Lyons, *Neural controlled differential equations for irregular time series*, in NeurIPS, 2020.

[107] Y. H. Kim, F. L. Lewis, and D. M. Dawson, *Intelligent optimal control of robotic manipulators using neural networks*, Automatica, 36 (2000), pp. 1355 – 1364.

[108] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[109] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907, (2016).

[110] M. R. Kirchner, M. J. Debord, and J. P. Hespanha, *A Hamilton–Jacobi formulation for optimal coordination of heterogeneous multiple vehicle systems*, in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 11623–11630.

[111] P. E. Kloeden and E. Platen, *Stochastic differential equations*, in Numerical Solution of Stochastic Differential Equations, Springer, 1992, pp. 103–160.

[112] M. A. Kramer, *Nonlinear principal component analysis using autoassociative neural networks*, AIChE journal, 37 (1991), pp. 233–243.

[113] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[114] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, *Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot*, Autonomous robots, 40 (2016), pp. 429–455.

[115] K. Kunisch, S. Volkwein, and L. Xie, *HJB-POD-based feedback design for the optimal control of evolution problems*, SIAM Journal on Applied Dynamical Systems, 3 (2004), pp. 701–722.

[116] O. Ladijzenskaia, *The mathematical theory of viscous incompressible fluid*, Gordon and, 667 (1969).

[117] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Transactions on Neural Networks, 9 (1998), pp. 987–1000.

[118] J. D. Lambert, *Numerical methods for ordinary differential systems*, vol. 146, Wiley New York, 1991.

[119] P. Lambrianides, Q. Gong, and D. Venturi, *A new scalable algorithm for computational optimal control under uncertainty*, Journal of Computational Physics, 420 (2020), p. 109710.

[120] D. Lee and C. J. Tomlin, *A Hopf-Lax formula in Hamilton–Jacobi analysis of reach-avoid problems*, IEEE Control Systems Letters, 5 (2020), pp. 1055–1060.

[121] D. Lee and C. J. Tomlin, *A Computationally Efficient Hamilton-Jacobi-based Formula for State-Constrained Optimal Control Problems*, arXiv e-prints, (2021), p. arXiv:2106.13440.

[122] K. Lee, N. A. Trask, and P. Stinis, *Machine learning structure preserving brackets for forecasting irreversible processes*, arXiv preprint arXiv:2106.12619, (2021).

[123] F. Lewis, D. Dawson, and C. Abdallah, *Robot Manipulator Control: Theory and Practice*, Control engineering, Marcel Dekker, 2004.

[124] A. Li, S. Bansal, G. Giovanis, V. Tolani, C. Tomlin, and M. Chen, *Generating robust supervision for learning-based visual navigation using Hamilton-Jacobi reachability*, in Learning for Dynamics and Control, PMLR, 2020, pp. 500–510.

[125] S.-H. Li, C.-X. Dong, L. Zhang, and L. Wang, *Neural canonical transformation with symplectic flows*, Physical Review X, 10 (2020), p. 021020.

[126] X. Li, *Simultaneous approximations of multivariate functions and their derivatives by neural networks with one hidden layer*, Neurocomputing, 12 (1996), pp. 327–343.

[127] F. Lin and R. D. Brandt, *An optimal control approach to robust control of robot manipulators*, IEEE Transactions on robotics and automation, 14 (1998), pp. 69–77.

[128] C. Livingston, *Knot theory*, vol. 24, Cambridge University Press, 1993.

[129] L. Lu, P. Jin, and G. E. Karniadakis, *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*, arXiv preprint arXiv:1910.03193, (2019).

[130] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*, Nature Machine Intelligence, 3 (2021), pp. 218–229.

[131] Y. Lu, C. Ma, Y. Lu, J. Lu, and L. Ying, *A mean-field analysis of deep resnet and beyond: Towards provable optimization via overparameterization from depth*, arXiv preprint arXiv:2003.05508, (2020).

[132] Y. Lu, A. Zhong, Q. Li, and B. Dong, *Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations*, in International Conference on Machine Learning, PMLR, 2018, pp. 3276–3285.

[133] C. Lubich, *From quantum to classical molecular dynamics: reduced models and numerical analysis*, European Mathematical Society, 2008.

[134] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in Proc. icml, vol. 30, 2013, p. 3.

[135] V. R. Makkapati, J. Ridderhof, P. Tsiotras, J. Hart, and B. van Bloemen Waanders, *Desensitized trajectory optimization for hypersonic vehicles*, in 2021 IEEE Aerospace Conference (50100), 2021, pp. 1–10.

[136] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Information Retrieval, 3 (2000), pp. 127–163.

[137] W. McEneaney, *A curse-of-dimensionality-free numerical method for solution of certain HJB PDEs*, SIAM Journal on Control and Optimization, 46 (2007), pp. 1239–1276.

[138] W. M. McEneaney, *Max-plus methods for nonlinear control and estimation*, Systems & Control: Foundations & Applications, Birkhäuser Boston, Inc., Boston, MA, 2006.

[139] W. M. McEneaney, A. Deshpande, and S. Gaubert, *Curse-of-complexity attenuation in the curse-of-dimensionality-free method for HJB PDEs*, in 2008 American Control Conference, IEEE, 2008, pp. 4684–4690.

[140] W. M. McEneaney and L. J. Kluberg, *Convergence rate for a curse-of-dimensionality-free method for a class of HJB PDEs*, SIAM Journal on Control and Optimization, 48 (2009), pp. 3052–3079.

[141] T. Meng, Z. Zhang, J. Darbon, and G. Karniadakis, *Sympocnet: Solving optimal control problems with applications to high-dimensional multiagent path planning problems*, SIAM Journal on Scientific Computing, 44 (2022), pp. B1341–B1368.

[142] X. Meng and G. E. Karniadakis, *A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems*, Journal of Computational Physics, 401 (2020), p. 109020.

[143] H. N. Mhaskar, *Neural networks for optimal approximation of smooth and analytic functions*, Neural computation, 8 (1996), pp. 164–177.

[144] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *Adaptive deep learning for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM Journal on Scientific Computing, 43 (2021), pp. A1221–A1247.

[145] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *QRnet: Optimal regulator design with LQR-augmented neural networks*, IEEE Control Systems Letters, 5 (2021), pp. 1303–1308.

[146] K. N. Niarchos and J. Lygeros, *A neural approximation to continuous time reachability computations*, in Proceedings of the 45th IEEE Conference on Decision and Control, Dec 2006, pp. 6313–6318.

[147] I. Omelyan, I. Mryglod, and R. Folk, *Symplectic analytically integrable decomposition algorithms: classification, derivation, and application to molecular dynamics, quantum and celestial mechanics simulations*, Computer Physics Communications, 151 (2003), pp. 272–314.

[148] D. Onken, L. Nurbekyan, X. Li, S. W. Fung, S. Osher, and L. Ruthotto, *A neural network approach for high-dimensional optimal control*, arXiv preprint arXiv:2104.03270, (2021).

[149] H. C. Öttinger, *Beyond equilibrium thermodynamics*, John Wiley & Sons, 2005.

[150] H. C. Öttinger and M. Grmela, *Dynamics and thermodynamics of complex fluids. ii. illustrations of a general formalism*, Physical Review E, 56 (1997), p. 6633.

[151] C. Parzani and S. Puechmorel, *On a Hamilton-Jacobi-Bellman approach for coordinated optimal aircraft trajectories planning*, Optimal Control Applications and Methods, 39 (2018), pp. 933–948.

[152] E. Parzen, *On estimation of a probability density function and mode*, The annals of mathematical statistics, 33 (1962), pp. 1065–1076.

[153] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep learning library*, Advances in neural information processing systems, 32 (2019), pp. 8026–8037.

[154] H. J. Pesch, *A practical guide to the solution of real-life optimal control problems*, Control and cybernetics, 23 (1994), p. 2.

[155] H. Qin, J. Liu, J. Xiao, R. Zhang, Y. He, Y. Wang, Y. Sun, J. W. Burby, L. Ellison, and Y. Zhou, *Canonical symplectic particle-in-cell method for long-term large-scale simulations of the Vlasov–Maxwell equations*, Nuclear Fusion, 56 (2015), p. 014001.

[156] T. Qin, K. Wu, and D. Xiu, *Data driven governing equations approximation using deep neural networks*, Journal of Computational Physics, 395 (2019), pp. 620–635.

[157] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Multistep neural networks for data-driven discovery of nonlinear dynamical systems*, arXiv preprint arXiv:1801.01236, (2018).

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

[159] A. Ranicki, *High-dimensional knot theory: Algebraic surgery in codimension 2*, Springer Science & Business Media, 2013.

[160] A. V. Rao, *A survey of numerical methods for optimal control*, Advances in the Astronautical Sciences, 135 (2009), pp. 497–528.

[161] C. Reisinger and Y. Zhang, *Rectified deep neural networks overcome the curse of dimensionality for nonsmooth value functions in zero-sum games of nonlinear stiff systems*, Analysis and Applications, 18 (2020), pp. 951–999.

[162] D. J. Rezende, S. Racanière, I. Higgins, and P. Toth, *Equivariant Hamiltonian flows*, arXiv preprint arXiv:1909.13739, (2019).

[163] R. Rico-Martinez, J. Anderson, and I. Kevrekidis, *Continuous-time nonlinear signal processing: a neural network based approach for gray box identification*, in Proceedings of IEEE Workshop on Neural Networks for Signal Processing, IEEE, 1994, pp. 596–605.

[164] R. Rico-Martinez, K. Krischer, I. Kevrekidis, M. Kube, and J. Hudson, *Discrete-vs. continuous-time nonlinear signal processing of cu electrodissolution data*, Chemical Engineering Communications, 118 (1992), pp. 25–48.

[165] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer, *An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments*, IEEE Robotics and Automation Letters, 3 (2018), pp. 1215–1222.

[166] I. Romero, *Thermodynamically consistent time-stepping algorithms for nonlinear thermomechanical systems*, International journal for numerical methods in engineering, 79 (2009), pp. 706–732.

[167] I. M. Ross and M. Karpenko, *A review of pseudospectral optimal control: From theory to flight*, Annual Reviews in Control, 36 (2012), pp. 182–197.

[168] V. R. Royo and C. Tomlin, *Recursive regression with neural networks: Approximating the HJI PDE solution*, arXiv preprint arXiv:1611.02739, (2016).

[169] A. Rucco, G. Notarstefano, and J. Hauser, *An efficient minimum-time trajectory generation strategy for two-track car vehicles*, IEEE Transactions on Control Systems Technology, 23 (2015), pp. 1505–1519.

[170] A. Rucco, P. Sujit, A. P. Aguiar, J. B. De Sousa, and F. L. Pereira, *Optimal rendezvous trajectory for unmanned aerial-ground vehicles*, IEEE Transactions on Aerospace and Electronic Systems, 54 (2017), pp. 834–847.

[171] L. Ruthotto and E. Haber, *Deep neural networks motivated by partial differential equations*, Journal of Mathematical Imaging and Vision, (2018).

[172] A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia, *Hamiltonian graph networks with ODE integrators*, arXiv preprint arXiv:1909.12790, (2019).

[173] M. Schmidt and H. Lipson, *Distilling free-form natural laws from experimental data*, science, 324 (2009), pp. 81–85.

[174] G. Schneider, P. F. Craigmile, and R. Herbei, *Maximum likelihood estimation for stochastic differential equations using sequential kriging-based optimization*, arXiv preprint arXiv:1408.2441, (2014).

[175] D. V. Schroeder, *An introduction to thermal physics*, 1999.

[176] X. Shang and H. C. Öttinger, *Structure-preserving integrators for dissipative systems based on reversible–irreversible splitting*, Proceedings of the Royal Society A, 476 (2020), p. 20190446.

[177] J. W. Siegel and J. Xu, *Approximation rates for neural networks with general activation functions*, Neural Networks, 128 (2020), pp. 313–321.

[178] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016), p. 484.

[179] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, Journal of Computational Physics, 375 (2018), pp. 1339 – 1364.

[180] S. Spedicato and G. Notarstefano, *Minimum-time trajectory generation for quadrotors in constrained environments*, IEEE Transactions on Control Systems Technology, 26 (2017), pp. 1335–1344.

[181] Y. Tang, L. Vázquez, F. Zhang, and V. Pérez-García, *Symplectic methods for the nonlinear Schrödinger equation*, Computers & Mathematics with Applications, 32 (1996), pp. 73–83.

[182] E. Todorov, *Efficient computation of optimal actions*, Proceedings of the national academy of sciences, 106 (2009), pp. 11478–11483.

[183] Y. Tong, S. Xiong, X. He, G. Pan, and B. Zhu, *Symplectic neural networks in Taylor series form for Hamiltonian systems*, arXiv preprint arXiv:2005.04986, (2020).

[184] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, and I. Higgins, *Hamiltonian generative networks*, in International Conference on Learning Representations, 2020.

[185] L. N. Trefethen and D. I. Bau, *Numerical linear algebra*, vol. 50, Siam, 1997.

[186] E. Trélat, *Contrôle optimal: théorie & applications*, vol. 36, Vuibert Paris, 2005.

[187] D. Turaev, *Polynomial approximations of symplectic dynamics and richness of chaos in non-hyperbolic area-preserving maps*, Nonlinearity, 16 (2002), pp. 123–135.

[188] P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, *Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics*, Neural Networks, 126 (2020), pp. 191 – 217.

[189] J. Wang, H. Sun, and S. Fiori, *A Riemannian-steepest-descent approach for optimization on the real symplectic group*, Mathematical Methods in the Applied Sciences, 41 (2018), pp. 4273–4286.

[190] S. Xiong, Y. Tong, X. He, C. Yang, S. Yang, and B. Zhu, *Nonseparable symplectic neural networks*, arXiv preprint arXiv:2010.12636, (2020).

[191] I. Yegorov and P. M. Dower, *Perspectives on characteristics based curse-of-dimensionality-free numerical approaches for solving Hamilton–Jacobi equations*, Applied Mathematics & Optimization, (2017), pp. 1–49.

[192] H. Yu, X. Tian, E. W, and Q. Li, *OnsagerNet: Learning stable and interpretable dynamics using a generalized Onsager principle*, arXiv preprint arXiv:2009.02327, (2020).

[193] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, *Deep sets*, Advances in neural information processing systems, 30 (2017).

[194] R. Zhang, J. Liu, Y. Tang, H. Qin, J. Xiao, and B. Zhu, *Canonicalization and symplectic simulation of the gyrocenter dynamics in time-independent magnetic fields*, Physics of Plasmas, 21 (2014), p. 032504.

[195] Z. Zhang, Y. Shin, and G. E. Karniadakis, *GFINNs: GENERIC formalism informed neural networks for deterministic and stochastic dynamical systems*, arXiv preprint arXiv:2109.00092, (2021).

[196] Y. D. Zhong, B. Dey, and A. Chakraborty, *Symplectic ODE-Net: Learning Hamiltonian dynamics with control*, in International Conference on Learning Representations, 2020.

[197] M. Zhou, J. Han, and J. Lu, *Actor-critic method for high dimensional static Hamilton–Jacobi–Bellman partial differential equations based on neural networks*, SIAM Journal on Scientific Computing, 43 (2021), pp. A4043–A4066.

[198] A. Zhu, P. Jin, and Y. Tang, *Deep Hamiltonian networks based on symplectic integrators*, arXiv preprint arXiv:2004.13830, (2020).

[199] ——, *Inverse modified differential equations for discovery of dynamics*, arXiv preprint arXiv:2009.01058, (2020).

[200] ——, *Approximation capabilities of measure-preserving neural networks*, arXiv preprint arXiv:2106.10911, (2021).

[201] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, *Beyond homophily in graph neural networks: Current limitations and effective designs*, Advances in neural information processing systems, 33 (2020), pp. 7793–7804.