HyperVerlet

A Deep Learning Method for Numerically Solving Initial Value Problems of Hamiltonian Systems

Anders Madsen & Frederik Baymler Mathiesen
Department of Computer Science, MI1012F21, 2021-06

Master's Thesis

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
http://www.aau.dk

AALBORG UNIVERSITY

STUDENT REPORT

Title:
HyperVerlet: a Deep Learning Method for Numerically Solving Initial Value Problems of Hamiltonian Systems

Theme:
Machine Intelligence

Project Period:
Fall Semester 2010

Project Group:
MI1012F21

Participant(s):
Anders Madsen
Frederik Baymler Mathiesen

Supervisor(s):
Bin Yang
Jilin Hu

Copies: 1

Page Numbers: 62

Date of Completion:
June 10, 2021

Abstract:

In this thesis, we propose HyperVerlet, a novel self-supervised deep learning method for numerically solving initial value problems of Hamiltonian systems. We provide theoretical proof of a lower local and global truncation error compared to the prevalent velocity Verlet. To gather empirical evidence, HyperVerlet is tested on dynamical systems where conservative quantities such as energy and total momentum is of high importance including an undamped spring-mass, an idealized pendulum, and the chaotic three-body spring-mass systems. Empirical evidence shows that HyperVerlet outperforms velocity Verlet, other hypersolvers, and in some cases perform on par with 4^{th}-order solvers. Depending on the choice of the neural network-based corrector, HyperVerlet has the capability of being both a symplectic and non-symplectic solver, which is a geometric property characterizing volume-preservation, making it ideal of long duration simulations.

Summary

In this thesis, we propose HyperVerlet, a novel self-supervised deep learning method for numerically solving Initial Value Problems of Hamiltonian systems. Hamiltonian systems are dynamical systems obeying Hamilton's equations, and Hamiltonian's equations are often not analytically solvable, thus requiring numerical solvers to approximate. An example of a system that cannot be solved analytically is the three-body spring-mass system, consisting of 3 point-masses and each pair of particles are connected by a spring. This system has been proven to exhibit deterministic chaos meaning that despite the uniqueness of the trajectory given initial conditions, small permutations in the initial condition can have tremendous impact on the trajectory, thereby implying that deterministic behavior is not equivalent with predictability. Tasks like these pose a challenge for neural networks, as they only approximate the fundamental laws of physic, like conservation of energy, which may lead to unfaithful simulations where energy can accumulate or dissipate as time progresses. For this reason, we propose a novel numerical hypersolver by the name HyperVerlet, which approximates the fundamental laws of physics better than existing hypersolvers.

HyperVerlet is a predictor-corrector architecture composed of a symplectic predictor, namely the velocity Verlet prevalent within molecular dynamics simulations due to its symplecticity, and a neural network-based corrector. HyperVerlet is trained by self-supervised learning, such that labels are self-contained within the data, and data can be generated on the fly by high-precision solvers. The trained solver is system-specific.

To show that HyperVerlet is superior to velocity Verlet, we present a theoretical proof showing that the truncation error of HyperVerlet is a constant factor lower than for velocity Verlet. We provide empirical evidence on three different dynamical systems, undamped spring-mass, idealized pendulum, and three-body spring-mass, that shows how HyperVerlet outperforms classical numerical methods like, Euler, Heun and velocity Verlet and state of the art hypersolvers like HyperEuler and HyperHeun, and for few cases perform on par with 4^{th}-order solvers.

To test the ability of HyperVerlet to generalize to variations in the input space, we conduct three experiments. The first experiment tests the impact of varying the step size during training, the second experiment tests how well HyperVerlet interpolates, conducted by randomly sampling to the mass and task specific parameters from the same distribution under training and testing, and the third experiment tests the extrapolation capabilities of HyperVerlet by sampling the mass and task specific parameters from different distributions for training and test datasets. Our generalization study shows that for some cases, HyperVerlet fails to conserve the energy, resulting in accumulation of energy and poor performance. We contribute this to the fact that the neural network, i.e. the corrector, used in the experiments are non-symplectic. We propose an alteration to the corrector making it symplectic. The extension is inspired by SympNets where layers are structured to be symplectic maps, thereby maintaining symplecticity through the entire architecture.

Different choices regarding the design of HyperVerlet has different impacts. To gain insight into the importance of different features, we conduct an ablation study, which is an alteration or removal of certain features to study their impact. We analyze the impact of loss functions, training methods, corrector architectures, and model inputs. The conclusion is that the training method, namely residual training, has the biggest impact followed by the choice of inputs. The study of different training methods for dynamical and interacting systems is of broader interest to the field as contemporary literature has no consensus on the superior method.

Contents

Preface

We would like to thank Morten Mattrup Smedskjaer, Tao Du, and Rasmus Christensen from the Oxide Glass Chemistry Group, Department of Chemistry and Bioscience at Aalborg University for sharing their chemistry and physics expertise with us, which gave us a lot of useful insight into the background and practical issues of molecular dynamics simulations including how they work with simulations. Furthermore, we would also like to thank them for their feedback on our ideas throughout the creation of this thesis. Likewise, we would like to thank Simon Mejlby Virenfeldt and Nikolaj Jensen Ulrik for helpful discussions surrounding the process, method, code, and thesis in general. Finally, we would like to extend our thanks to Nicklas Hansen for his valuable feedback on our thesis.

Code for the implementation our method and the experiments of this thesis, and visualizations of the simulations are available at `https://github.com/Zinoex/hyperverlet/`.

Aalborg University, June 10, 2021

Anders Madsen
<amads15@student.aau.dk>

Frederik Baymler Mathiesen
<fmathi16@student.aau.dk>

Chapter 1

Introduction

NeuralODEs combine the continuous dynamics of Ordinary Differential Equations (ODEs) with the expressivity of neural networks to form a continuous depth model that is very appropriate for modeling continuous time series (Chen et al., 2018). One kind of continuous time series is a dynamical system, which is a system whose behavior is a function of time determined by its dynamics and whose state is uniquely defined by a set of variables. Following the work on NeuralODEs, there has been an increased interest in bridging the gap between deep learning and dynamical systems, both to model dynamical systems using neural networks and using the methodology of dynamical system to improve neural networks (Cranmer et al., 2020; Greydanus et al., 2019; Poli et al., 2020b; Shen et al., 2020; Sanchez-Gonzalez et al., 2019). Dynamical systems are a useful tool for studying motion in systems whose behavior are mechanical in nature, making it useful in a wide variety of systems ranging from a simple pendulum over molecular dynamics to the motion of planetary orbits.

The class of dynamical systems, we study in this thesis, are Hamiltonian systems, which means they obey Hamilton's equations. Hamilton's equations are two ODEs implying that the dynamics of the system are deterministic, i.e. there is a unique solution given the initial conditions of the system (Ott, 2002). However, even for simple systems like three celestial bodies orbiting each other, there are no analytical solution, thus requiring numerical methods. Additionally, this system has been proved by Henri Poincaré to exhibit deterministic chaos. A system is chaotic if its time evolution is sensitive small perturbations in initial conditions, also known as the butterfly effect, implying that deterministic behavior is not equivalent with predictability.

Despite the immense diversity in tasks, they all share the same underlying laws of physics. As an example, the effect of gravity is the same for almost all objects, no matter if the object is a planet orbiting a sun or whether its a single particle flying through space. This realization has lead to a trail of research in machine learning trying to find physics priors across different tasks, which may lead to better generalization across models (Greydanus et al., 2019; Cranmer et al., 2020; Ummenhofer et al., 2019; Sanchez-Gonzalez et al., 2019).

An example of such physical prior is conservation of energy, which states that the total energy of an isolated system will remain constant. This means that if we consider a swinging bob in an ideal pendulum system, i.e. no frictional energy loss, the bob should always return to its initial condition. Despite the deceptive simplicity of the problem, it poses a great challenge for neural networks without physical priors. This problem emerges as the neural network is merely learning an approximation of the physics directly from data, and thereby prevents it from learning the exact physical laws. As a consequence of this approximation, over time the system will start to either dissipate or accumulate energy, hence the bob will either converge at resting position or accelerate to infinity.

Historically, the choice of Initial Value Problem (IVP) solver has been a key aspect for incorporating the physical priors (Verlet, 1967). An IVP solver is a numerical method to solve ODEs of which Hamilton's equations, a formulation of classical mechanics for conservative systems exhibiting symplecticity or volume-preservation, is an important class. The solver is the simulator or engine behind the temporal evolution of the dynamical system. They are used across a wide range of applications including LAMMPS, a molecular dynamics framework, Matlab, a general purpose engineering tool, OpenGym, a toolkit for developing and comparing reinforce-

ment learning algorithms, and many others. Even though the choice of solver has a huge impact on the output, as we will demonstrate, the solver choice is largely overlooked, as a key example the cartpole environment in OpenGym uses a simple Euler integrator[1], which is notoriously inaccurate and unstable. The issue of treating the choice of solver as a simple hyperparameter was pointed out in (Poli et al., 2020b). However, they treat the general class of ODEs and thus do not attempt to incorporate the physical priors into their proposed hypersolvers.

In this thesis, we will introduce a novel hypersolver by the name HyperVerlet. The HyperVerlet uses the symplectic velocity Verlet, commonly used in Molecular Dynamics (MD) simulations, as its base solver and extend it with a neural network to create a predictor-corrector architecture. The HyperVerlet is trained by self-supervised learning where the label is self-contained within the data, in our case by a temporal split, and thus avoiding the tedious task of labeling data and allowing data to be generated on the fly through high-precision solvers. We provide emperical evidence that the HyperVerlet outperforms state of the art hypersolvers like the HyperEuler and HyperHeun (Shen et al., 2020; Poli et al., 2020b) on three independent dynamical systems. Furthermore, we provide theoretical proof for the local truncation error of HyperVerlet.

The thesis is organized as follows. Chapter 2 will outline prerequisite theory including different flavors of mechanics and various integration algorithms used for predicting the trajectories of these mechanics. A literature review is found in Chapter 3, portraying contemporary machine learning techniques within the intersection of physics, dynamical systems, and deep learning. Chapter 4 describes our novel approach, HyperVerlet, how it incorporates physical priors into a deep learning model, architectural choices, training procedure, and theoretical proofs of properties including truncation errors. We test our method across a range of methods, which we will detail in Chapter 5. The results of our experiments are found in Chapter 6 including a baseline comparison and a generalization and ablation study. Chapter 7 will discuss different considerations one should bear in mind when incorporating physical priors into deep learning. Finally, Chapter 8 will summarize and conclude the thesis.

[1]Source found at `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

Chapter 2

Theory

2.1 Classical mechanics

To understand the formulation and impact of the HyperVerlet method and the chosen experiments, we must develop a core understanding of classical mechanics. We begin with a treatise on the Newtonian formulation of mechanics as described by Newton in his famous *Philosophiæ Naturalis Principia Mathematica*, or Mathematical Principles of Natural Philosophy, of 1687. Our work on all formulations of classical mechanics is based on (Newton, 1687) and (Taylor, 2005).

Starting with an arbitrary object, we have two fundamental quantities: position q and mass m. Usually, the position described by 2- or 3-dimensional Cartesian coordinates, and in this case, we can decompose q into (x, y) or (x, y, z) depending on the number of dimensions. The first and second derivative of position with respective to time are velocity and acceleration respectively

$$v = \dot{q} = \frac{dq}{dt} \tag{2.1}$$

$$a = \ddot{q} = \dot{v} = \frac{d^2q}{dt^2} \tag{2.2}$$

Here, the number of dots above a character denotes the order of derivatives of that quantity with respect to time. The position is relative to a frame of reference, i.e. we can select any coordinate system without changing the dynamics of the system. For Newtonian mechanics, the frame of reference must also be inertial meaning that the frame of reference is not accelerating. Imagine a person standing on the earth and another person standing a train passing by. We can consider the physics from the perspective of either, i.e. both can be a frame of reference, but if the train is accelerating, the person on the train will not be an inertial frame of reference while the person on the earth will. In this work, we consider only inertial frames of reference with zero velocity, and only objects of constant mass. Additionally, we only treat dynamics for linear and synchronous time, i.e. we do not account for relativity as discovered by Einstein. This constraint is sufficient as many physical phenomena like pendulums, springs, and the movement of atoms, which only travel at speeds much less than the speed of light.

With these core definitions and constraints, we can define Newton's laws of motion. The Newtonian formulation of mechanics treats dynamics as the relationship between the motion of objects and forces acting upon the objects.

Definition 1 (Newton's first law)

Every body perseveres in its state of rest, or of uniform motion in a right line, unless it is compelled to change that state by forces impressed thereon (Newton, 1687).

♢

This obtuse formulation says that a change in velocity, i.e. an acceleration, is induced by forces acting upon the the object. The mathematically equivalent expression is

$$F = 0 \iff \frac{dv}{dt} = 0 \tag{2.3}$$

where F denotes the sum of forces acting upon the object and v is the velocity of the same object. Note here that an object can be both the idealized point-mass approximation and aggregations of atoms like a pendulum bob, planets traveling through space or all particles in a molecular dynamics system. If the sum of forces of internal forces, e.g. between particles, is zero then the acceleration of the object will be zero unless acted upon by an external force.

Definition 2 (Newton's second law)

The alteration of motion is ever proportional to the motive force impressed; and is made in the direction of the right line in which that force is impressed (Newton, 1687).

♢

This law establishes the direct linear relationship between forces acted upon an object and the change in motion, or momentum p, of the same object.

$$F = \dot{p} \tag{2.4}$$

Momentum is the product of mass and velocity, $p = m \cdot v$, and since we assume constant mass, the rate of change of momentum only depends on the rate of change of velocity, i.e. acceleration, yielding the famous equation

$$F = m \cdot a \tag{2.5}$$

Definition 3 (Newton's third law)

To every action there is always opposed an equal action; or the mutual actions of two bodies upon each other are always equal, and directed to contrary parts (Newton, 1687).

♢

This law is of particular importance to our molecular dynamics experiments as the motion of a particle not only depends on the other particles but the particle itself also impacts the motion of all other particles. When any pair of particles in these systems interact, the sum of forces of the particles acting upon each other is zero.

An implication of the third law of motion is the conservation of momentum, i.e. the sum of all momenta in an isolated system remains zero. If we consider a system where a light object A and a heavy object B collide, conversation of momentum $\dot{p}_A - \dot{p}_B = 0$ implies that the lighter object A will have to accelerate much harder than B to match the momentum of the heavier object B.

While Newton's three laws sufficiently describe the non-relativistic world, it quickly becomes intractable with larger number of objects interacting with each other. Lagrangian mechanics is an alternative but equivalent formulation that is easier to scale to larger systems.

2.1.1 Lagrangian mechanics

Lagrangian mechanics, invented by Joseph-Louis Lagrange in 1788, revolves around the concept of kinetic and potential energy, and this formulation is based around configuration space rather than individual forces, which allows easier scaling to larger systems. Configuration space is the concept of high-dimensional coordinates to describe the position of all objects in a physical system. If we consider a particle system of N particles in 3-dimensional space, the configuration space will be $3N$ dimensional. The evolution of a system in configuration space is called a trajectory.

To define this flavor of mechanics, we need two central concepts: Lagrangian and action.

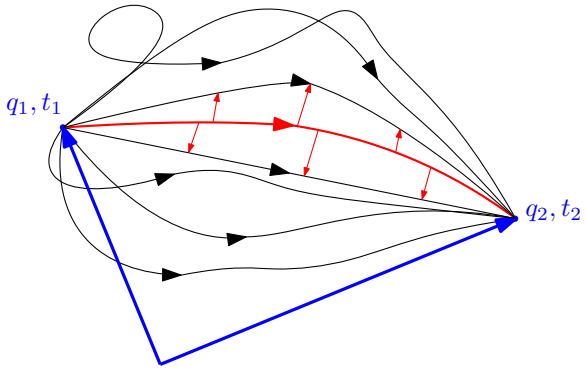


Figure 2.1: Principle of least action where the edge conditions are (q_1, t_1) and (q_2, t_2). The true trajectory is the red trajectory and any change along the entire path will change action and thus not satisfy the principle of least (stationary) action.

Definition 4 (Lagrangian) The Lagrangian $L : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}_+ \to \mathbb{R}$ for non-relativistic systems is the difference between kinetic energy T and potential energy V where D is the size of the configuration space.

$$L(q, \dot{q}, t) = T(\dot{q}) - V(q) \tag{2.6}$$

where the potential energy $V(q)$ depend on the system being modeled and the kinetic energy is

$$T(\dot{q}) = \frac{1}{2} m \dot{q}^2 \tag{2.7}$$

♢

Definition 5 (Action) The action $\mathcal{S}$ is a functional from a configuration space trajectory from time t_1 to time t_2 to the integral of the Lagrangian for this trajectory with boundary conditions $q_1 = q(t_1)$ and $q_2 = q(t_2)$

$$\mathcal{S}[q(t)] = \int_{t_1}^{t_2} L(q(t), \dot{q}(t), t) \, dt \tag{2.8}$$

♢

To derive the equations of motion for Lagrangian mechanics, we use Hamilton's principle, which states that the action of the true trajectory is stationary. An example of a true trajectory is shown in Figure 2.1 where any different trajectory will have a different action due to some deviation along any subsegment of the path. If we consider an arbitrary function $f(t)$, we can find a stationary point, if such a point exists, by equating the derivative with zero, $\dot{f}(t) = 0$. Equivalently, we can find a stationary trajectory of a functional by equating its derivative with zero using calculus of variations, i.e. calculus for functionals.

$$\frac{\delta S[q(t)]}{\delta q(t)} = 0 \tag{2.9}$$

To solve this equation, assume we add a small perturbation $\varepsilon(t)$ to the trajectory $q(t)$ with the boundary conditions that $\varepsilon(t_1) = \varepsilon(t_2) = 0$. The change in action $\delta S[q(t)]$ wrt. position $q(t)$ is equal to $S[q(t) + \varepsilon(t)] - S[q(t)]$ as the perturbation ε approaches zero. For brevity we omit t as a parameter for $q(t)$ and $\varepsilon(t)$ in the following derivation.

$$\delta S[q] = \int_{t_1}^{t_2} L(q + \varepsilon, \dot{q} + \dot{\varepsilon}, t) - L(q, \dot{q}, t) \, dt = \int_{t_1}^{t_2} \varepsilon \cdot \frac{\partial L(q, \dot{q}, t)}{\partial q} + \dot{\varepsilon} \cdot \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \, dt = 0 \tag{2.10}$$

Doing integration by parts, we get

$$\delta S[q] = \left[\varepsilon \cdot \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \right]_{t_1}^{t_2} + \int_{t_1}^{t_2} \left(\varepsilon \cdot \frac{\partial L(q, \dot{q}, t)}{\partial q} - \varepsilon \cdot \frac{d}{dt} \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \right) dt = 0 \tag{2.11}$$

Since the boundary conditions of $\varepsilon(t)$ are zero, the first term becomes zero.

$$\delta S[q] = \int_{t_1}^{t_2} \varepsilon \cdot \left(\frac{\partial L(q, \dot{q}, t)}{\partial q} - \frac{d}{dt} \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \right) dt = 0 \tag{2.12}$$

The remaining statement must be equal to zero by Hamilton's principle invariant of the choice of $\varepsilon(t)$, which can only be the case if

$$\frac{\partial L(q, \dot{q}, t)}{\partial q} - \frac{d}{dt} \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} = 0 \tag{2.13}$$

This equation is known as the Euler-Lagrange equation and is the equation of motion for Lagrangian mechanics. To find the trajectory in configuration space for both single and multibody systems, we find the kinetic and potential energy and compute the solution to this second-order differential equation.

We can observe the equivalence between Newtonian and Lagrangian mechanics for non-relativistic, conservative systems by solving the Euler-Lagrange equation for $T(\dot{q}) = \frac{1}{2} m \dot{q}^2$. The conservative property signify that the system has no loss of energy, which implies that all forces are conservative, and conservative forces can by definition be written as the negative gradient of the potential energy function wrt. position.

$$\frac{\partial L(q, \dot{q}, t)}{\partial q} = \frac{d}{dt} \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}} \tag{2.14}$$

$$-\frac{\partial V(q)}{\partial q} = \frac{d}{dt} \frac{\partial}{\partial \dot{q}} \frac{1}{2} m \dot{q}^2 \tag{2.15}$$

$$F = ma \tag{2.16}$$

Thus, we can derive Newton's second law of motion from Lagrangian mechanics for non-relativistic, conservative systems.

2.1.2 Hamiltonian mechanics

While conservation of energy is a property of Newtonian mechanics, it is non-trivial to derive. For Lagrangian mechanics, the property is easier to prove and it forms the basis of Hamiltonian mechanics, invented by William Rowan Hamilton in 1833.

To prove conservation of energy, we can from the definition of the Lagrangian show that the partial derivative wrt. velocity is the momentum

$$\frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}_i} = \frac{\partial T(\dot{q}_i)}{\partial \dot{q}_i} = m_i \dot{q}_i = p_i \tag{2.17}$$

Additionally, we observe from the Euler-Lagrange equation for any coordinate i that

$$\frac{\partial L(q, \dot{q}, t)}{\partial q_i} = \frac{d}{dt} \frac{\partial L(q, \dot{q}, t)}{\partial \dot{q}_i} = \frac{d}{dt} p_i = \dot{p}_i \tag{2.18}$$

With these definitions, we can show how the Lagrangian changes over time with p and $\dot{q}$, called the Legendre transformation, and with this show that we have conservation of energy.

$$\frac{dL}{dt} = \sum_i \frac{\partial L}{\partial q_i}\dot{q}_i + \sum_i \frac{\partial L}{\partial \dot{q}_i}\ddot{q}_i + \frac{\partial L}{\partial t} \tag{2.19}$$

$$= \sum_i \dot{p}_i\dot{q}_i + \sum_i p_i\ddot{q}_i + \frac{\partial L}{\partial t} \tag{2.20}$$

$$= \frac{d}{dt}\left(\sum_i p_i\dot{q}_i\right) + \frac{\partial L}{\partial t} \tag{2.21}$$

The second term on the right hand side of Equation 2.21 is for many systems zero, and thus, we can compute a time invariant property for Lagrangian systems. This quantity is called the Hamiltonian and is denoted H.

$$\frac{d}{dt}\left(\sum_i p_i\dot{q}_i - L\right) = 0 \tag{2.22}$$

$$\frac{d}{dt}H(q,p,t) = 0 \tag{2.23}$$

For systems with a time-independent mapping between the generalized coordinates $q_1, \ldots, q_n$ and Cartesian coordinates, i.e. $H(q,p) = H(q,p,t)$, it is possible to prove that the Hamiltonian is the sum of kinetic and potential energy; that is the total energy (we refer to (Taylor, 2005) for proof). Thus, the energy of Lagrangian and Hamiltonian systems without explicit time-dependence is constant, and by the equivalence between Newtonian and Lagrangian mechanics, they also benefit from conservation of energy.

The state of Lagrangian systems without explicit time-dependence is captured by the positions and velocities of all bodies, which is evident by the Lagrangian being a function of both. Similarly, in Equation 2.23, we see that the Hamiltonian is a function both position and momentum, which implies that the state of a Hamiltonian system is captured by the combination of the two quantities. If we consider a system of N particles in 3-dimensional space as we did with Lagrangian mechanics, the coordinates for describing the state of this system in a Hamiltonian perspective is $6N$-dimensional as we need capture both the position and momentum. This type of coordinate is called a canonical coordinate and the vector space of canonical coordinates is called the phase space; the combination of configuration and momentum space.

With a initial coordinate in phase space, we need an equation to describe its trajectory as time evolves, that is, we need an equation of motion. For Hamiltonian systems, the dynamics is governed by two equations: one for positions and one for momenta. To derive the equation of motion for momenta, we start by differentiating the Hamiltonian wrt. position and using Equation 2.17 and 2.18 to obtain the force.

$$\begin{aligned}\frac{\partial H}{\partial q_i} &= \frac{\partial}{\partial q_i}\left(\sum_i p_i\dot{q}_i - L\right) \\ &= p_i\frac{\partial \dot{q}_i}{\partial q_i} - \left(\frac{\partial L}{\partial q_i} + \frac{\partial L}{\partial \dot{q}_i}\frac{\partial \dot{q}_i}{\partial q_i}\right) \\ &= p_i\frac{\partial \dot{q}_i}{\partial q_i} - \left(\dot{p}_i + p_i\frac{\partial \dot{q}_i}{\partial q_i}\right) \\ &= -\dot{p}_i\end{aligned} \tag{2.24}$$

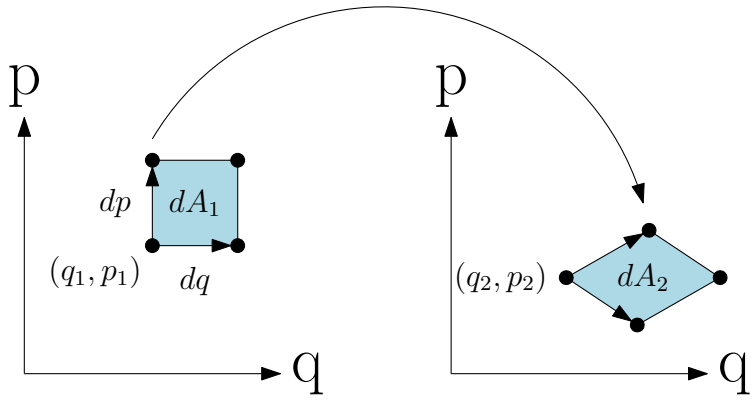


Figure 2.2: An area preserving transformation $dA_1 = dA_2$ of a quadrilateral from time t_1 to time t_2.

Similarly, we can the derive the equation of motion for position by differentiating the Hamiltonian wrt. momentum.

$$\begin{aligned}\frac{\partial H}{\partial p_i} &= \frac{\partial}{\partial p_i}\left(p_i\dot{q}_i - L\right) \\ &= \left(\frac{\partial p_i}{\partial p_i}\dot{q}_i + p_i\frac{\partial \dot{q}_i}{\partial p_i}\right) - \frac{\partial L}{\partial \dot{q}_i}\frac{\partial \dot{q}_i}{\partial p_i} \\ &= \dot{q}_i + p_i\frac{\partial \dot{q}_i}{\partial p_i} - p_i\frac{\partial \dot{q}_i}{\partial p_i} \\ &= \dot{q}_i\end{aligned} \tag{2.25}$$

Collectively, these two equations are called Hamilton's equations. We can combine Hamilton's equations into a single equation by letting $z = (q, p)$ be the canonical coordinate and taking the gradient of the Hamiltonian wrt. z. Here, I_N denotes the $N \times N$ identity matrix, and N is the number of coordinates in configuration space.

$$\dot{z} = \begin{bmatrix} 0 & I_N \\ -I_N & 0 \end{bmatrix} \cdot \nabla_z H = J \cdot \nabla_z H \tag{2.26}$$

A key difference between Lagrangian and Hamiltonian mechanics is that the equations of motion for Lagrangian mechanics are N seconder-order differential equations and for Hamiltonian mechanics, the equations of motion are $2N$ first-order differential equations. In many cases, this makes Hamiltonian systems easier to solve.

An important characterizing property of Hamiltonian system is the incompressibility of the phase space fluid, or symplecticity; a property asserted in Liouville's theorem. To understand this property imagine a 2-dimensional phase space as shown in Figure 2.2 and a rectangle with corners $(q_1, p_1), (q_1 + dq, p_1), (q_1, p_1 + dp), (q_1 + dq, p_1 + dp)$ at time t_1 where q_1, p_1 denote position and momentum respectively and dq, dp denote a small change in their respective variables. The area dA_1 occupied by this rectangle is $dA_1 = dq \cdot dp$. At a later time t_2, each of the four coor-

dinates have shifted according to Hamilton's equations with the new parallelogram occupying an area dA_2. Incompressibility implies that the two areas are equal, that is $dA_1 = dA_2$.

Formally, a fluid, in this case the phase space fluid, must be incompressible if and only if the divergence of the velocity field is zero. For a Hamiltonian system with a velocity field $\dot{z} = (\dot{q}, \dot{p})$ of the $2N$-dimensional phase space fluid, the divergence is defined as

$$\nabla \cdot \dot{z} = \sum_{i=1}^{N} \left(\frac{\partial \dot{q}_i}{\partial q_i} + \frac{\partial \dot{p}_i}{\partial p_i} \right) \tag{2.27}$$

Exploiting that the velocity field is described by Hamilton's equations, we arrive at

$$\nabla \cdot \dot{z} = \sum_{i=1}^{N} \left(\frac{\partial}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial}{\partial p_i} \frac{\partial H}{\partial q_i} \right) = 0 \tag{2.28}$$

Thus, the phase space fluid is incompressible, and we see that symplecticity is a geometric property about area preservation. Symplecticity is closely related to conservation of energy in the context of numerical integrators.

Since conservation of energy is a core aspect of Lagrangian and Hamiltonian mechanics, it is evident they are better suited for defining conservative systems. This excludes systems like non-idealized pendulums and other frictional systems, but it includes many interesting problems on macroscopic, i.e. cosmic, and microscopic, i.e. atomic or molecular, scale. If desired, one *can* apply Lagrangian and Hamiltonian mechanics to frictional systems.

2.2 Numerical Integration for ODEs

Hamilton's equations are multivariate first-order ODEs describing the dynamics or the derivative of the trajectory, but for practical applications, we want to recover the trajectory by solving the differential equation given initial conditions. This problem is called an Initial Value Problem (IVP) (Süli, 2003), and Figure 2.3 is an illustration of such a problem. To define the IVP problem, we need a definition of Lipschitz continuity.

Definition 6 (Lipschitz continuity) A function is Lipschitz continuous in x if, for any two timestamps $t_1, t_2 \in [t_0, \infty]$, there exists a real number $K > 0$ such that

$$\|f(x(t_1), t_1) - f(x(t_2), t_2)\| \leq K\|x(t_1) - x(t_2)\| \tag{2.29}$$

Thus, a Lipschitz continuous function has a bounded rate of change.

♢

Definition 7 (Initial Value Problem) Assume a function $f(x, t) : \mathbb{R}^n \times [t_0, \infty] \to \mathbb{R}^n$ is Lipschitz continuous in x and continuous in t. Then given an initial state $x_0 \in \mathbb{R}^n$, the initial value problem is a first-order ODE where a solution satisfies $x(t_0) = x_0$.

$$\frac{dx(t)}{dt} = f(x(t), t) \tag{2.30}$$

♢

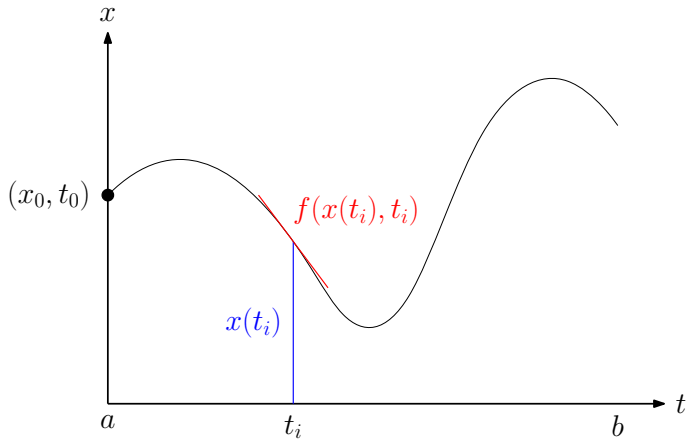


Figure 2.3: An IVP with known initial conditions (x_0, t_0). Since, the function $x(t)$ is unknown, we need to recover the specific trajectory from only the gradient $f(x(t), t)$ and the initial conditions (x_0, t_0).

While an analytical solution of the ODE with x as a function of time is desirable, it is often not achievable. To solve such problems, numerical integration allows approximating the solution to the first-order ODE given the initial conditions. Such numerical integrations rely on partitioning an interval into a series of discrete steps, i.e. a partition P of a closed, bounded interval $[a, b]$ is a finite, ordered set of points (Taylor, 2012)

$$P = \{a = t_0 < t_1 < \cdots < t_n = b\} \tag{2.31}$$

The partitioning methods define two families of solvers: fixed-step and adaptive solvers. While adaptive solvers partition the time interval on-the-fly, the fixed-step solvers precompute equal-width subintervals, i.e the step size $h = t_{i+1} - t_i$ for all $0 \leq i < n$. We only focus on fixed-step solvers.

Because the gradient is an instantaneous, continuous function and numerical integrations rely on temporal discretizations, the methods are prone to deviate from the exact trajectory. The stepwise error introduced is called the local truncation error τ_n and is usually quantified as an asymptotic power $O(h^{p+1})$ of the step size h (Süli, 2003).

$$\tau_n = x(t_n) - (x(t_{n-1}) + h \cdot \psi(t_{n-1}, x(t_{n-1}), h, f)) \tag{2.32}$$

where ψ denotes the step function and $x(t_n)$ denotes the true x at time t_n. A solver with a local truncation error of $O(h^{p+1})$ is called a p^{th}-order solver.

The global truncation error e_n induced across a number of steps is also an asymptotic function $O(h^p)$, and for single-step methods, i.e. methods that only track a single, latest state, the relationship between the local and global truncation is a power of one if the step function is Lipschitz continuous (Süli, 2003). The global truncation error is defined as

$$e_n = x(t_n) - x_n \tag{2.33}$$

where x_n is computed by n times repeated application of a numerical ODE solver. To reduce the truncation error, a solution is to divide the time interval into smaller subintervals, and if the recovered trajectory approaches the true trajectory as the number of subintervals goes to infinity then the solver is called convergent; a desirable property related to the stability of the solver.

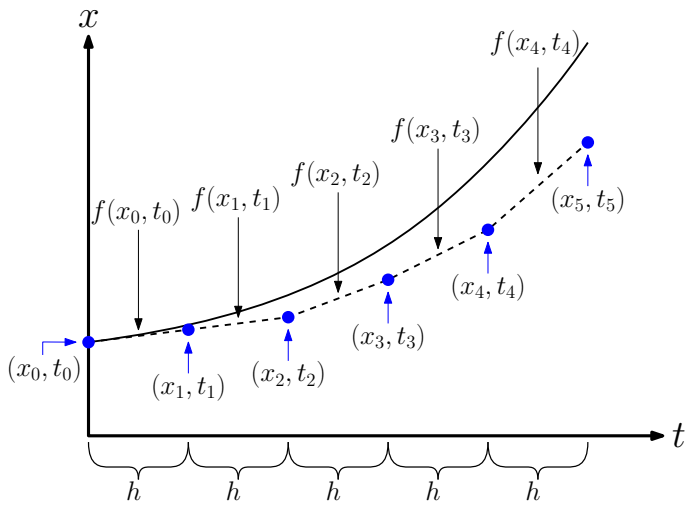


Figure 2.4: Geometric illustration of Euler method. The solid line illustrates the unknown curve we wish to approximate, and the dashed line illustrates an approximation using Euler Method. $f(x_n, t_n)$ denotes the tangent line or gradient to point (x_n, t_n).

However, smaller step sizes require increasingly many computations, reducing the problem to a speed/accuracy trade-off. Furthermore, as the step size decreases floating point rounding error becomes relevant (Fässler, 2019), which implies that there exists an absolute limit in accuracy.

The following sections will explore different algorithms for computing the solution to an IVP along with their benefits and short-comings.

2.2.1 Euler method

Euler method is the earliest and simplest numerical method for solving ordinary differential equations (Fässler, 2019). To derive Euler method assume that the derivative of $x(t)$, that is $f(x(t), t)$, is constant. Then the exact solution to the IVP could be written as $x(t) = x_0 + f(x_0, t_0)(t - t_0)$. The idea behind Euler method is to relax the assumption of a globally constant derivative to only locally constant derivatives, i.e. a constant derivative within each subinterval of the partitioning. Starting from the initial configuration (t_0, x_0), the method approximates the next point (t_1, x_1) by taking a small step of size h in the direction of the gradient and repeating the process for the next subinterval, see Figure 2.4. We can formulate this process as

$$x_{n+1} = x_n + h \cdot f(x_n, t_n) \qquad t_{n+1} = t_n + h \tag{2.34}$$

Euler method is convergent meaning that as the step size approaches zero $h \to 0$ the approximated solution approaches the exact solution $x[n] \to x(t)$, assuming infinite numerical precision. However, Euler method is a 1^{st}-order method and notoriously unstable implying the need for very small time steps to produce usable trajectories.

2.2.2 Heun method

To address the shortcomings of the Euler method, better methods have been invented. One such method is the Heun method, also known as improved Euler method (Fässler, 2019). One of the weaknesses in the Euler method is the basic assumption of a locally constant derivative, which

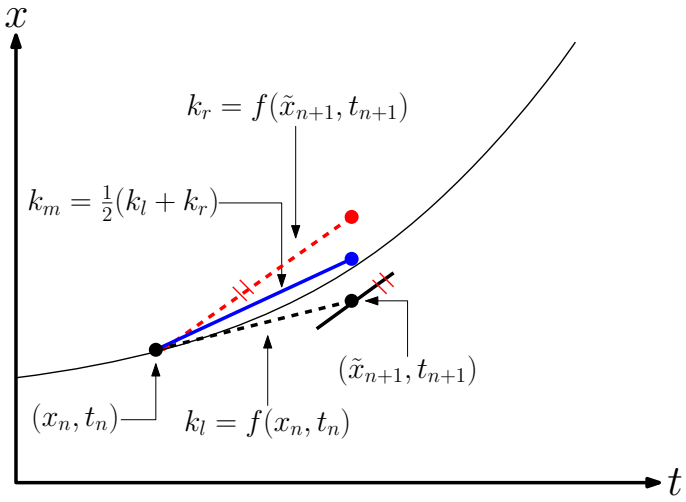


Figure 2.5: Geometric illustration of Heun's method. Black dashed line illustrate the tangent at the left end point k_l, where the solid black and red dashed line illustrate the tangent to the approximated right end point k_r, and the blue dot being the average of the two namely k_m.

requires a sufficiently small step size. However in practice, errors accumulate over time and the prediction and the true solution will eventually deviate significantly.

Looking at Figure 2.4, we can observe that if the trajectory is curving up, i.e. the derivative is positive, the Euler method will always underestimate the x_n value at time t_n. The opposite relationship applies for a downward trending curve, i.e. a negative gradient, in which case the Euler method always will overestimate the value x_n. Heun method alleviates this over- and underestimation problem by considering the tangent to the solution curve at both ends of the interval. It does so by using the Euler method to make an intermediate prediction for the right end point $\tilde{x}_{n+1}$ such that it can compute the gradient to this point k_r. Figure 2.5 illustrates how the tangent to the left end point k_l underestimates the true trajectory, and the tangent to the right end point k_r approximated by the Euler method overestimates, but when taking the average of the these two tangents k_m, the approximation is closer to the true solution. The step computation is as follows:

$$\begin{aligned} k_l &= f(x_n, t_n) & k_r &= f(\tilde{x}_{n+1}, t_{n+1}) \\ t_{n+1} &= t_n + h & k_m &= \frac{1}{2}(k_l + k_r) \\ \tilde{x}_{n+1} &= x_n + hk_l & x_{n+1} &= x_n + hk_m \end{aligned} \tag{2.35}$$

The Heun method is more stable than the Euler method and is a 2^{nd}-order method, and thus, the method allows bigger time steps without sacrificing the accuracy.

2.2.3 Runge-Kutta method

While the Euler method is notoriously unstable, the Heun method improves this considerably by using two rather than one tangent line. Therefore, the intuitive next step would be to add more tangents for improving the accuracy and stability of the prediction, and this approach is called the Runge-Kutta methods (Fässler, 2019).

One of the most commonly used Runge-Kutta methods is the 4^{th}-order Runge-Kutta (RK4). To compute RK4, the same approach as Heun is used with some modifications. Rather than only

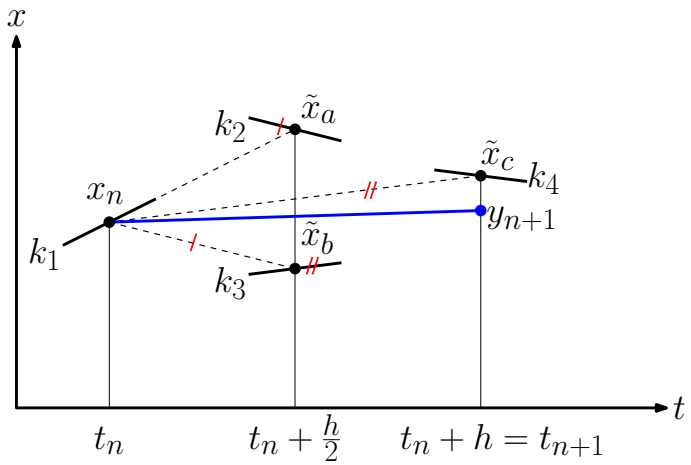


Figure 2.6: Geometric illustration of a 4th-order Runge-Kutta solver.

evaluating the tangent at either end of the step interval $[t_n, t_n + h]$, the method also evaluates two tangent lines at the midpoint $t_n + \frac{h}{2}$, namely k_2 and k_3 as shown in Figure 2.6, and uses these intermediate values to compute the tangent of k_4. When all the tangents are computed, a weighted average of the gradients based on the trapezoidal rule is used to compute the final estimation x_{n+1}. The computation is as follows:

$$\begin{aligned} k_1 &= f(x_n, t_n) & \tilde{x}_a &= x_n + \frac{h}{2}k_1 \\ k_2 &= f(\tilde{x}_a, t_n + \frac{h}{2}) & \tilde{x}_b &= x_n + \frac{h}{2}k_2 \\ k_3 &= f(\tilde{x}_b, t_n + \frac{h}{2}) & \tilde{x}_c &= x_n + hk_3 \\ k_4 &= f(\tilde{x}_c, t_n + h) & x_{n+1} &= x_n + \frac{h}{3}\left(\frac{1}{2}k_1 + k_2 + k_3 + \frac{1}{2}k_4\right) \end{aligned} \tag{2.36}$$

2.2.4 Velocity Verlet

Extending beyond the class of traditional single-step IVP solvers is the class of symplectic integrators of which the most prevalent is the velocity Verlet algorithm. This algorithm is the velocity form of the Störmer–Verlet algorithm (Verlet, 1967), which is named after Loup Verlet who applied it to molecular dynamics simulations of Lennard-Jones molecules. While the algorithm is named after Verlet, he only rediscovered it in the context of molecular dynamics (Hairer et al., 2003). It has previously been discovered by Jean Delambre for astronomy and Carl Störmer for aurora borealis, showing its usefulness and wide applicability. The velocity form was discovered in 1982, again within the context of molecular dynamics (Swope et al., 1982).

An necessary assumption to apply the algorithm is that the initial position and velocity is known. The algorithm works by computing the forces acting on the system at time t_n and from this compute the acceleration, $a(q_n) = F(q_n)/m$, and combine the acceleration $a(q_n)$ and the velocity v_n to advance the configuration space coordinate q_{n+1} as can be seen in Equation 2.37. Note that acceleration is computed only from the position and not velocity, a condition of conservative Hamiltonian systems that the algorithm utilizes to ensure symplecticity. With the

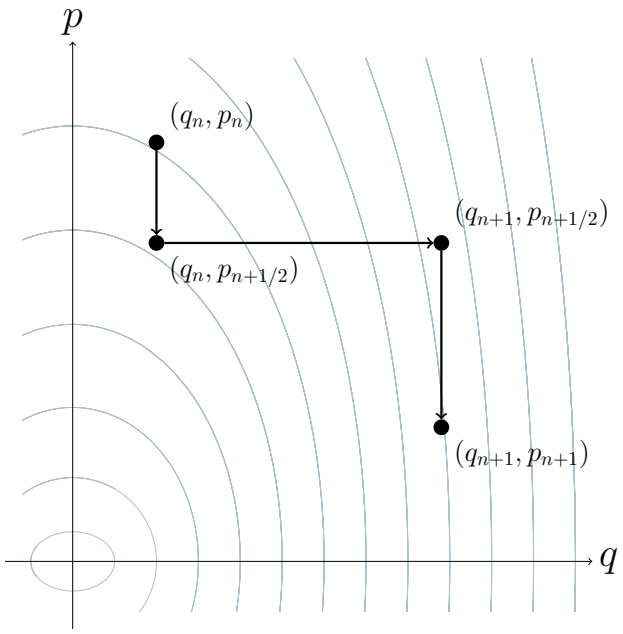


Figure 2.7: A single step of the velocity Verlet integrator in a 2-dimensional phase space. The visualization show the 3-step form of velocity Verlet, i.e. updating $p \rightarrow q \rightarrow p$. Note the relationship $p = mv$, which means the velocity Verlet can be reformulated in terms of momentum.

next position q_{n+1}, it is then possible to compute the velocity v_{n+1} at time t_{n+1}, by taking the average of the acceleration at time t_n and t_{n+1} as shown in Equation 2.38.

$$q_{n+1} = q_n + h \cdot v_n + \frac{h^2}{2} \cdot a(q_n) \tag{2.37}$$

$$v_{n+1} = v_n + h \cdot \frac{a(q_n) + a(q_{n+1})}{2} \tag{2.38}$$

Alternatively, velocity Verlet can be written in a 3-step form as the equations below. The geometric interpretation of the 3-step form can be seen in Figure 2.7.

$$v_{n+1/2} = v_n + \frac{h}{2} \cdot a(q_n) \tag{2.39}$$

$$q_{n+1} = q_n + h \cdot v_{n+1/2} \tag{2.40}$$

$$v_{n+1} = v_{n+1/2} + \frac{h}{2} \cdot a(q_{n+1}) \tag{2.41}$$

The usefulness of velocity Verlet is a result of its symplecticity, time reversibility, and computational efficiency. We analyzed the symplectic property in Section 2.1.2 but for numerical integrators, symplecticity has a profound impact of approximating the true energy by $O(h^p)$ for a pth-order solver (Hairer et al., 2003). Time reversibility is the ability to apply the same algorithm to simulate a system backwards in time, that is $h < 0$. For velocity Verlet, this property is evident from the symmetry of the 3-step form.

Since velocity Verlet is a core aspect of this thesis, we derive its local truncation error and by extension its order. We note that any function $x(t_n)$ can be written as a Taylor expansion

truncated after a number of terms, in this case four

$$x(t_n) = x(t_{n-1}) + \dot{x}(t_{n-1})h + \frac{1}{2}\ddot{x}(t_{n-1})h^2 + O(h^3) \tag{2.42}$$

Starting from the definition of local truncation error as Equation 2.32, we split the derivation into position and velocity; we will call the local truncation errors τ_n^q and τ_n^v respectively. Substituting $q(t_n)$ in the definition of the local truncation error with its Taylor expansion noting that velocity and acceleration is the first- and second-order derivative of position wrt. time, we can reduce the expression to an asymptotic power of the time step.

$$\begin{aligned}
\tau_n^q &= q(t_n) - q(t_{n-1}) - h \cdot v(t_{n-1}) - \frac{h^2}{2} \cdot a(t_{n-1}) \\
&= q(t_{n-1}) + v(t_{n-1})h + \frac{h^2}{2}a(t_{n-1}) + O(h^3) - q(t_{n-1}) - v(t_{n-1})h - \frac{h^2}{2} \cdot a(t_{n-1}) \\
&= O(h^3)
\end{aligned} \tag{2.43}$$

With this derivation, we can conclude that the local truncation error is $O(h^3)$ and the order is 2 wrt. position for velocity Verlet. We can do a similar derivation for velocity noting once again that acceleration is the first-order derivative of velocity wrt. time. Additionally, it is important that we can substitute $h^2/2 \cdot \dot{a}(t_{n-1})$ with $h/2 \cdot (a(t_n) - a(t_{n-1})) + O(h^3)$ by its Taylor expansion.

$$\begin{aligned}
\tau_n^v &= v(t_n) - v(t_{n-1}) - h \cdot \frac{a(t_{n-1}) + a(t_n)}{2} \\
&= v(t_{n-1}) + a(t_{n-1})h + \frac{h^2}{2}\dot{a}(t_{n-1}) + O(h^3) - v(t_{n-1}) - h \cdot \frac{a(t_{n-1}) + a(t_n)}{2} \\
&= a(t_{n-1})h + \frac{h^2}{2}\dot{a}(t_{n-1}) + O(h^3) - h \cdot \frac{a(t_{n-1}) + a(t_n)}{2} \\
&= a(t_{n-1})h + h \cdot \frac{a(t_n) - a(t_{n-1})}{2} + O(h^3) - h \cdot \frac{a(t_{n-1}) + a(t_n)}{2} \\
&= O(h^3)
\end{aligned} \tag{2.44}$$

Thus, we can conclude velocity Verlet is a 2^{nd}-order integrator with respect to velocity also.

2.2.5 Forest-Ruth Symplectic Integrators

Like non-symplectic integrators, there exists higher-order symplectic integrators with the goal of achieving higher precision with less computation (Ruth, 1983; Forest & Ruth, 1990). Two such, 3^{rd}- and 4^{th}-order, were discovered by Ronald Ruth in 1983, but only published later in collaboration with Etienne Forest, which is why we call these Forest-Ruth symplectic integrators. The symplectic 4^{th}-order solver follows the same pattern as the 3-step form of the velocity Verlet algorithm with alternating between updating the position and momentum. Within a single time step, we index this alternating behavior with i, and the computation is as follows with $(\tilde{q}_0, \tilde{v}_0) = (q_n, v_n)$ and $(\tilde{q}_4, \tilde{v}_4) = (q_{n+1}, v_{n+1})$

$$\tilde{q}_1 = \tilde{q}_0 + \frac{1}{2(2 - 2^{1/3})} \cdot \tilde{v}_0 h \tag{2.45}$$

$$\tilde{v}_1 = \tilde{v}_0 + \frac{1}{2 - 2^{1/3}} \cdot a(\tilde{q}_1)h \tag{2.46}$$

$$\tilde{q}_2 = \tilde{q}_1 + \frac{1 - 2^{1/3}}{2(2 - 2^{1/3})} \cdot \tilde{v}_1 h \tag{2.47}$$

$$\tilde{v}_2 = \tilde{v}_1 - \frac{2^{1/3}}{2 - 2^{1/3}} \cdot a(\tilde{q}_2) h \tag{2.48}$$

$$\tilde{q}_3 = \tilde{q}_2 + \frac{1 - 2^{1/3}}{2(2 - 2^{1/3})} \cdot \tilde{v}_2 h \tag{2.49}$$

$$\tilde{v}_3 = \tilde{v}_2 + \frac{1}{2 - 2^{1/3}} \cdot a(\tilde{q}_3) h \tag{2.50}$$

$$\tilde{q}_4 = \tilde{q}_3 + \frac{1}{2(2 - 2^{1/3})} \cdot \tilde{v}_3 h \tag{2.51}$$

$$\tilde{v}_4 = \tilde{v}_3 \tag{2.52}$$

By noticing the symmetry in equations around Equation 2.48, it becomes evident that this method also has time symmetry as velocity Verlet; a desired property. However, some of the coefficients are negative, which can be interpreted as a step backwards in time, which is counter-intuitive and a reason for its dislike in many applications. We choose to use a 4^{th}-order Forest-Ruth as to obtain ground truth trajectories for its high precision.

2.2.6 Comparison

To summarize, the two key distinguishing features of single-step integrators are their order, which measures how fast they converge with smaller time steps, and whether they are symplectic, which is a geometric property of area/volume preservation. The latter is an important characteristic for applying these methods to Hamiltonian systems. Table 2.1 compares all the various integrators on these two parameters as well as their learnability, i.e. whether they use machine learning to improve the performance. The HyperEuler and HyperHeun integrators are described in Section 3.5, and the HyperVerlet integrator, our method, is described in Chapter 4.

Table 2.1: Compare integrators by their order, symplecticity, and neural network-based corrector. The symplecticity of the HyperVerlet integrator depends on the choice of a neural corrector.

Integrator	Order	Symplectic	Learnable
Euler	1	✗	
HyperEuler	1	✗	✓
Heun	2	✗	
HyperHeun	2	✗	✓
4^{th}-order Runge-Kutta	4	✗	
Velocity Verlet	2	✓	
HyperVerlet	2	✓/ ✗	✓
4^{th}-order Forest-Ruth	4	✓	

Chapter 3

Literature Review

3.1 Machine Learning for physical simulations

Despite extensive research in machine learning for physical simulations, many open questions remain including novel types of simulations, the efficacy of machine learning simulations, and frameworks for future research. Some of the most difficult properties of physical simulations for machine learning to capture, and by extension one of the biggest inhibitors for its wider adoption, are the conservation of quantities. Examples include conservation of energy, momentum, angular momentum, and mass. The following section is a review of recent progress in machine learning for physical simulations, and we will emphasize their goal, method, and limitations including their conservative properties.

A field greatly benefiting from learned physical simulations is computer vision where a learned simulator can automate years of engineering and design efforts. One such method is Graph-Network Based Simulators (GNS) (Sanchez-Gonzalez et al., 2020), which uses graph neural networks to learn the dynamics of rigid and deformable solids, incompressible fluids, and granular material. The simulator is a graph network (Battaglia et al., 2018) where a particle approximation of the material under study represents nodes in a graph, and the edges represent the local neighborhood of interaction. The benefit of using graph networks is the permutation equivariance of the particles and their interactions, and the generalization of the simulator to larger domains with more particles and obstacles. While the method conserves the mass of the system, it fails to learn the conversation of internal momentum, i.e. momentum accounting for the gravity applied.

To improve upon GNS specifically for incompressible fluids, i.e. liquids, Fluid Graph Networks (FGN) (Li & Farimani, 2021) take inspiration from physics to dividing the inference into three separate steps: advection or the movement of a larger body of mass, collision between particles, and pressure to account for density differences. Due to the pressure step, FGN outperform GNS on density accuracy measures, but otherwise the methods are of similar performance. The most significant difference between the methods is the inference speed where FGN offer 8x the inference speed by having significantly fewer weights and message-passing steps.

Another type of simulation benefiting from machine learning for improving inference speed is computational fluid dynamics (Kochkov et al., 2021). To simulate fluids, the Naiver-Stokes equations can be solved numerically by partitioning the simulation domain into a grid where the trade-off between accuracy and computational feasibility is controlled by the grid spacing. (Kochkov et al., 2021) propose a method for simulating coarser grids for faster simulations while retaining the accuracy by interpolating on the grid cells using machine learning methods. The architecture includes a divergence operator to enforce local conservation of momentum.

For many of types of physical simulations, a benefit of machine learning is the ability to approximate physical quantities where exact computations are expensive and slow. Molecular dynamics is no exception where *ab initio* simulations, i.e. quantum mechanics simulations, are computationally heavy but necessary to analyze certain behaviors (Botu & Ramprasad, 2015). A solution to this problem is presented by (Botu & Ramprasad, 2015) where machine learning is used to predict the force exerted on all particles rather than using Density Functional Theory for exact computations. One interesting aspect of this method is the quantification of the stepwise

predicability of the dynamics with the machine learning model, i.e. a quantification of whether the machine learning model can predict the next step. If the stepwise predictability is less than a threshold, the algorithm reverts to the more expensive quantum mechanics computation to maintain an accurate computation.

All the aforementioned simulations are of very specific types, but classical mechanics (see Section 2.1) are general and as a result, one can define general simulations for which similarly general machine learning models can be defined. This is the goal of Hamiltonian Neural Network (HNN) (Greydanus et al., 2019) and Lagrangian Neural Network (LNN) (Cranmer et al., 2020) that predict the Hamiltonian and Lagrangian respectively, and from this scalar quantity compute the acceleration of all bodies by backpropagation. Hamiltonian ODE Graph Network (HOGN) (Sanchez-Gonzalez et al., 2019) is very similar to Hamiltonian Neural Networks where the Hamiltonian is predicted on a graph, specifically with a Graph Network (Battaglia et al., 2018). The structure is very appropriate for molecular dynamics simulations due to the variable number of nodes and permutation equivariance. HNN and LNN make use of an Euler integrator (see Section 2.2.1), which is notoriously unstable and non-symplectic, while HOGN uses an RK4 integrator (see Section 2.2.3), which is more stable but remains non-symplectic.

There has been a rising interest in applying machine learning for molecular dynamics as evident by recent developments in tools. Frameworks for molecular dynamics including deterministic and learned potential energy and force functions, periodic boundary conditions, space partitioning, data structures, system initialization, and NVE/NVT/Nosé-Hoover simulations has been developed for Jax (Schoenholz & Cubuk, 2020) and Pytorch (Doerr et al., 2021). Since both Jax and Pytorch are frameworks for deep learning, we may readily reap the benefits of improving their solver.

3.2 Neural Ordinary Differential Equation

Since classical mechanics are often represented by differential equations, contemporary ODE methods for neural networks are promising for modeling simulations of these. The paper (Chen et al., 2018) first derived how a residual layer of infinitesimally small width can be interpreted as an ordinary differential equation. This interpretation allows the application of ODE solvers, which can be explicitly tuned for a speed/accuracy trade-off, to compute the solution of the IVP parameterized by the neural network. Treating neural networks as ODE are also appropriate for modeling time series as the integration range can be variable. However, being an early application of ODE solvers to neural networks, this method has several issues including performance being heavily integrator-dependent, batching is non-trivial, and reconstruction of the trajectory on reserve-mode ODE is not guaranteed due to lack of symmetry of the integrator and the adjoint method (Gholaminejad et al., 2019).

Since NeuralODE can encode the dynamics of time series, (Rubanova et al., 2019) applied the method to encoding time series data into latent space. The method uses NeuralODE for evolving the latent representation through time and a Recurrent Neural Network (RNN) for updating the latent vector with new samples, and the advantage of this method compared to only using a RNN is a better performance for irregularly-sampled data.

Many problems in physics and economics cannot be represented by ODEs but instead requires Stochastic Differential Equations (SDEs); especially if we cannot observe the finest granularity or isolate the system under study. To this end, (Li et al., 2020) generalize NeuralODEs to

SDEs by using SDE solvers and store a Brownian tree efficiently for backward integration.

Additional extensions of NeuralODE include ODEs on nodes in static graphs with Graph Neural Networks (GNNs) for interactions (Poli et al., 2020a) and higher order regularization for smoother functions resulting in fewer function evaluations (Kelly et al., 2020).

3.3 Symplectic Neural Networks

Other researchers have discovered the benefit of applying the symplectic structure to neural networks to simulations of Hamiltonian systems and even for latent representations. Variational Integrator Networks (VINs) (Saemundsson et al., 2020) is one such method where a physical system is encoded into a latent vector, which is split into two parts: position and momentum. The model then use a Verlet integrator to construct the trajectory in the latent space and finally decode the representation to the physical domain. This encode-process-decode architecture can be intepreted as augmenting NeuralODE with a symplectic integrator and constraining to a non-variable step size. With the symplectic integrator, the architecture conserves energy and momentum in latent space, which in turn improves interpretability and accuracy for long-term predictions, and allows data-efficient learning. The performance of VINs is impressive for simpler systems like idealized mass-spring systems and pendulums, even in the presence of noisy samples, however, the generalizability to multibody and chaotic systems such as double pendulums or molecular dynamics is unexplored.

An approach learning directly on the physical domain is Symplectic Recurrent Neural Networks (SRNNs) (Chen et al., 2020) where deficiencies of HNN are identified and ameliorated. Specifically as we saw in our literature review of HNN, it uses an Euler integrator, which is non-symplectic and unstable, and thus, it fails to learn conservation of energy and predict longer trajectories. SRNN solves the issue by replacing the Euler integrator with a velocity Verlet integrator, which is both symplectic and stable in a larger region (see Section 2.2.4). Additionally, SRNN learns from noisy data by leveraging the multistep, or recurrent, training for its averaging behavior. Like most models, the method suffers when modeling very stiff systems (see Section 5.1) and requires explicit augmentation. Very similar to SRNN, TaylorNet (Tong et al., 2021) introduces symplecticity to HNN by replacing the Euler integrator, but with a 4^{th}-order Forest-Ruth (FR4) integrator (see Section 2.2.5) instead of velocity Verlet. TaylorNet also constrains the structure of the underlying network to a linear combination of symmetric non-linear terms. Each term is a linear-activation-linear block where the two linear layers are the transpose of each other and the activation is a term of the Taylor polynomial, hence the name. A major issue with this approach is the requirement of a fixed domain size, i.e. the network is trained on the dynamics of our solar system but will not work for any other solar systems with a different number of celestial bodies or celetial bodies of different masses.

The symplectic structure is not only useful for simulating systems with conservation of energy but also for accurate control of physical systems. Symplectic ODE-Net (Zhong et al., 2020) learns to accurately control inherently unstable system such as balancing a double pendulum in an upwards position by modeling the dynamics as a Hamiltonian Neural Network controlled through shaping the desired potential and kinetic energy functions. The method is not strictly a symplectic network as it does not employ a symplectic integrator but an RK4 integrator, which only remains sufficient for short trajectories with small step sizes. However, the system is a closed feedback-loop system, and thus, the deficient integrator has a lesser impact.

For symplectic systems, we can also reject the idea of using an integrator by letting a neural network predict the position and momentum as a function of time as done by (Mattheakis et al., 2020). They train this network with a novel loss function incorporating the derivative of network and the Hamiltonian for learning the symplectic structure, and show that this network is symplectic as the loss approaches zero and the time horizon approaches infinity. However, the method has several disadvantages where the major limitation is the lack of generalization. The cause is that the network is trained to specific initial conditions and has more parameters than samples. Additionally, the lack of a comparison to state of the art severely limits our understanding of whether this direction is worth continued pursuit.

Another method rejecting the idea of integrators for neural networks is SympNets (Jin et al., 2020) where the layers are structured to be symplectic maps to maintain the symplectic property of the simulator. The symplectic layers model both the ODEs and the symplectic integration, which collectively outperform HNN with a symplectic integrator. A limitation of this method is the generalizability as task specific parameters such as masses, gravity, and length are encoded into the learned parameters, and lack of scalability for multibody system as it treats each object as having its own dynamics rather than applying the same dynamics to all particles.

3.4 Physics-inspired optimization

While Lagrangian and Hamiltonian mechanics are useful for describing physical systems, its applicability extend beyond the physical realm (Betancourt, 2017; Jordan, 2018). In the context of convex optimization theory, momentum methods has received great attention in the last decade (Sutskever et al., 2013), and it has also been studied from the perspective of Lagrangian and Hamiltonian mechanics (Jordan, 2018). Starting with Stochastic Gradient Descent (SGD) where the update step $\theta \leftarrow \theta - \alpha \nabla L(\theta)$ can be interpreted as an Euler integration in high-dimensional configuration space with α being the step size. Adding acceleration or momentum to SGD, the optimization becomes similar to following a configuration space trajectory. Indeed, (Wibisono et al., 2016) show that in the continuous-time limit, Nesterov momentum Accelerated Gradient Descent (AGD) can be derived from Lagrangian mechanics yielding the interpretation that it is a discretization method of the underlying dynamics. Additionally, from this variational perspective, the authors show that a naive discretization, i.e. Euler integration, is inherently unstable at the convergence rate of the underlying differential equation, giving more credit to the momentum methods.

A theoretical result showing the benefit of applying classical mechanics to convex optimization is derived by (Wibisono et al., 2016), but lack empirical evidence of its efficacy. This is partly due to difficult discretization originating from the second-order differential Euler-Lagrange equation. Similarly to physical systems, the discretization problem is solved by converting the Lagrangian formulation to a Hamiltonian formulation, which reduces from one second-order differential equation to two first-order differential equations and can be solved with symplectic integrators yielding faster optimization than AGD (Betancourt et al., 2018). Despite the impressive results, symplectic optimization is not the solution to solve all optimization problems. E.g. its performance severely degrades for stochastic objectives and error analysis is complicated by the symplectic integrators.

To combat the issues with symplectic optimization for stochastic objectives, (Jordan, 2018) show how the Hamiltonian mechanics can be replaced by Langevin mechanics, namely under-

damped Langevin diffusion, instead. Langevin mechanics incorporate stochastic behavior by its SDE formulation. Additionally, (Jordan, 2018) generalize symplectic optimization to non-convex optimization via a momentum perturbation to escape saddle points.

Since gradient-based optimization can be solved by symplectic integrators (Betancourt et al., 2018), improvements of these integrators has the potential of improving optimization methods too. If this improvement is achieved by incorporating a neural network into the integrator, we essentially have a neural network, the integrator, optimizing another neural network, the model.

3.5 Deep Integrators

Recently, efforts have been made to extend traditional integrators, or IVP solvers, with deep learning. Namely, two concurrent works with very similar approaches: Deep Euler method (Shen et al., 2020) and hypersolvers (Poli et al., 2020b).

Starting with the Deep Euler method, it augments an Euler integrator with a neural network with the goal of improving the inaccurate base solver, increasing the step size for faster computation times, and allowing easier integration of stiff ODEs.

Definition 8 (Deep Euler method) Given an IVP as in Definition 7 and a feed-forward neural network $g(\theta, x_n, t_n, t_{n+1})$, the evolution is given by

$$x_{n+1} = x_n + h \cdot f(x_n, t_n) + h^2 \cdot g(\theta, x_n, t_n, t_{n+1}) \tag{3.1}$$

The neural network minimizes the residual R, which is the difference between the predicted next value x_{n+1} using an Euler integrator and the ground truth $x(t_{n+1})$, usually obtained by a high-precision solver, scaled by the time step size.

$$R = \frac{1}{h^2}\left[x(t_{n+1}) - (x_n + h \cdot f(x_n, t_n))\right] \tag{3.2}$$

$$\mathcal{L} = \frac{1}{N}\sum_{k=1}^{N} \|R(x_n, t_n, t_{n+1}) - g(\theta, x_n, t_n, t_{n+1})\|_1 \tag{3.3}$$

♢

With this particular method, they achieve results significantly better than a regular Euler integrator and generalize the method to higher-order single-step methods by defining the update step and residual as

$$x_{n+1} = x_n + h \cdot \psi(x_n, t_n, h) + h^{p+1} \cdot g(\theta, x_n, t_n, t_{n+1}) \tag{3.4}$$

$$R = \frac{1}{h^{p+1}}\left[x(t_{n+1}) - (x_n + h \cdot \psi(x_n, t_n, h))\right] \tag{3.5}$$

where $\psi(x_n, t_n, h)$ is any single-step update function and p is the order of the base solver. E.g. for Euler method, the update function is $\psi(x_n, t_n, h) = f(x_n, t_n)$, and for Heun's method, it is $\psi(x_n, t_n, h) = [f(x_n, t_n) + f(x_n + h \cdot f(x_n, t_n), t_{n+1})]/2$. Neither Deep Euler or Deep Heun are compared to higher-order methods like RK4, which would have been interesting given its prevalence and efficiency. One caveat of using a neural network corrector is the fact that the order of the solver does not change but only changes by a constant factor, but the stability is vastly improved compared to an Euler integrator.

The hypersolvers, HyperEuler and HyperHeun, are the exact same methods as Deep Euler and Deep Heun respectively but developed independently[1] (Poli et al., 2020b). These methods are derived from the perspective of NeuralODE to reduce the necessary number of function evaluations and achieve wall-clock speed ups. While you can train the NeuralODE and the hypersolver jointly, it presents certain challenges, and therefore, the hypersolvers only explore independent training. This paper also shows the Pareto efficiency of hypersolvers over their traditional base solvers. Finally, the paper discusses two alternative training methods to residual fitting: trajectory fitting and adversarial training. Trajectory fitting is where the hypersolver is used to predict a trajectory that is measured against a ground truth trajectory using Mean Squared Error (MSE). The issue of this method is that the errors accumulate along the trajectory and introduces noise into the training process. The second training method, adversarial training, is an interesting perspective because the goal is to learn stiff ODEs by exploiting the weaknesses in the traditional solvers.

Our work is heavily influence by theses two papers and their ideas, and our method is an extension from non-symplectic to symplectic solvers exploiting the predictor-corrector architecture of these deep integrators.

[1] Henceforth, we will use the names HyperEuler and HyperHeun to refer to these equivalent methods.

Chapter 4

Method

Coarsening a simulation of physical systems can be done by either coarsening the configuration space or using larger time steps for the solver. Since spatial coarsening is problem dependent and time coarsening is problem invariant, we choose to focus on larger time steps without sacrificing accuracy. To this end, our novel method is the extension of deep solvers (Shen et al., 2020) and hypersolvers (Poli et al., 2020b) to a symplectic base solver, allowing larger time steps for the simulation of Hamiltonian systems.

The method is a predictor-corrector architecture where the symplectic base solver is the predictor, and the neural network is the corrector. We visualize the architecture in the context of an abstract system in Figure 4.1. The system can be either a known system like a double pendulum or a molecular system, or a learned system like NeuralODE (Chen et al., 2018), HNN (Greydanus et al., 2019), or LNN (Cranmer et al., 2020). The solver query the system for the acceleration a given the position q and mass m as described by Hamilton's equations (see Equation 2.26).

The core distinction between hypersolvers and our HyperVerlet is the choice of the base solver where hypersolvers use an arbitrary single-step solver and our method explicitly requires a symplectic solver. The solver restriction is inspired by physical simulations like molecular dynamics and cosmic simulations and is a necessity for the improved performance and conservation of energy, and allowing for longer and more stable simulations.

Mathematically, we treat the output of the predictor on Figure 4.1, a velocity Verlet integrator, as an intermediate output

$$\tilde{q}_{n+1} = q_n + h \cdot v_n + \frac{h^2}{2} a(q_n) \tag{4.1}$$

$$\tilde{v}_{n+1} = v_n + h^2 \frac{a(q_n) + a(\tilde{q}_{n+1})}{2} \tag{4.2}$$

Denoting a neural network g_θ, the corrector on Figure 4.1, we use this to correct the intermediate output

$$\begin{bmatrix} \hat{q}_{n+1} \\ \hat{v}_{n+1} \end{bmatrix} = \begin{bmatrix} \tilde{q}_{n+1} \\ \tilde{v}_{n+1} \end{bmatrix} + h^3 \cdot g_\theta(v_n, a(q_n), \tilde{v}_{n+1}, a(\tilde{q}_{n+1}), m, \Gamma) \tag{4.3}$$

where $v_n, a(q_n)$ denotes the derivative of the position and velocity before the predictor update, $\tilde{v}_{n+1}, a(\tilde{q}_{n+1})$ is the derivative of the position and velocity after the update, m is the mass, and Γ is time invariant, task specific parameters required for computing the derivatives. Note that the inputs are different from deep- and hypersolvers since both architectures take as input the time t_n and the prior state (q_n, v_n). However, since the prior position q_n is not multiplied by the step size h in Equation 4.1, it will not contribute to the truncation error, thus we exclude it. The reason for excluding t_n is that the truncation error is not affected by the explicit time but rather the rate of change of the state of the system, which is related to the definition of a stiff system (see Section 5.1) and also why we include these. Finally, we include task specific parameters Γ that are necessary for computing the derivatives as they convey information about the dynamics of the system and by extension the truncation error. Examples of Γ are the idle length and spring constant for a spring-mass system or the gravity and length of pendulum (see Section 5).

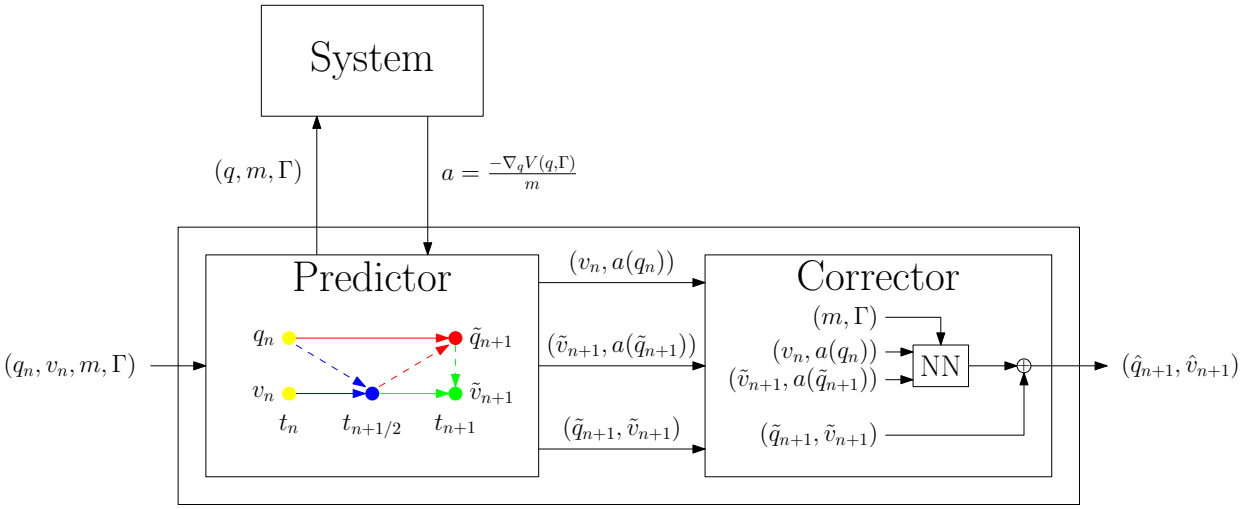


Figure 4.1: The architecture of our HyperVerlet IVP solver and an abstract system to simulate given initial conditions (q_0, p_0). The predictor is the velocity Verlet algorithm, which computes an intermediate state $(\tilde{q}_{n+1}, \tilde{v}_{n+1})$ by querying the system for the acceleration at q_n and $\tilde{q}_{n+1}$. The corrector, a neural network, computes the residual and adds it to the intermediate state, yielding the output $(\hat{q}_{n+1}, \hat{v}_{n+1})$. m and Γ are the mass and task specific parameters respectively. Collectively, the predictor and corrector is the solver.

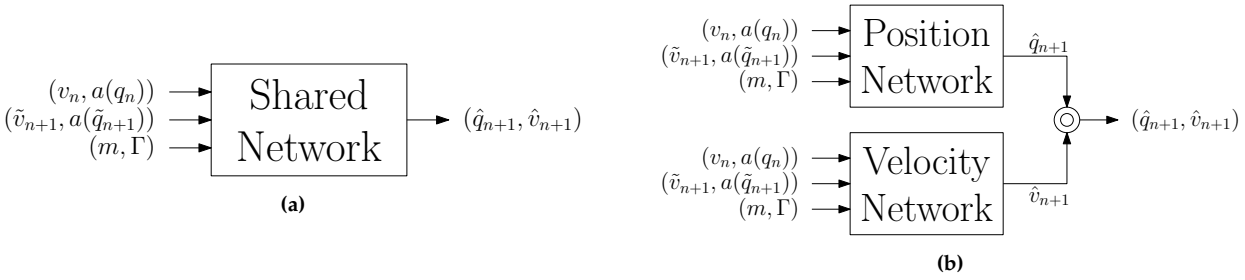


Figure 4.2: Shared vs individual networks for position $q \in \mathbb{R}^d$ and velocity $v \in \mathbb{R}^d$. **(a)** shows a shared network with a $2d$-dimensional vector output for both position and velocity. **(b)** shows two individual networks with a d-dimensional output for position and velocity respectively. The $\circ$ operator denotes a concatenation.

4.1 Corrector architecture

While any neural network obeying the inputs and outputs of Equation 4.3 can be a corrector, we explore the impact of certain architectures. An important architectural choice is whether to have two networks, one for position and one for velocity, or to share weights. The benefits of shared feature extraction are fewer weights, hopefully resulting in a faster training, and higher accuracy attributed to common features. However, since the update functions for position and velocity deviate significantly, we expect the error patterns to be equally different resulting in a deficient feature extractor if the weights are shared.

The architecture for the g_θ network is task specific. E.g. for a single pendulum, the size of the q and p vectors are 1, but for a double pendulum, the size is 2, which results in the input and output of network being of different sizes and the network, if fully-connected, not being transferable between tasks. Additionally, the error patterns are different depending on the task thus requiring retraining for each system.

For certain tasks, models beyond fully-connected exhibit desirable traits. One such class is Graph Networks (GNs), which has been proven useful for particle dynamics due to their permutation equivariance, i.e. any permutation of the input results in the same permutation of the output. Mathematically, permutation equivariance can be described by assuming an adjacency matrix A and a permutation of the adjacency matrix σ, then

$$GN(\sigma(A)) = \sigma(GN(A)) \tag{4.4}$$

Additionally, GNs share the weights between all nodes, i.e. particles, and thus apply the same error correction dynamics to all particles, and allows altering the simulation size, i.e. number of particles, between training samples and at inference time.

4.2 Training method

Since the method is a temporal step function, it is possible to construct a trajectory only based on the initial conditions and backpropagate through time. This multi-step training procedure is called recurrent training by (Chen et al., 2020) as the network effectively can be considered a type of recurrent neural networks. The alternative is non-recurrent or single-step training, which avoids accumulating errors and reduces noise in the training procedure but is less robust to noisy training data. There is no consensus in literature about which of the two methods is best; (Kipf et al., 2018) and (Chen et al., 2020) consider multi-step training beneficial because it penalizes degenerate step functions and improves robustness to noise, while (Sanchez-Gonzalez et al., 2020) believes that multi-step training reduces the generalization of the model because single-step training imposes a stronger Markovian biases that physical domains have.

Similar to (Poli et al., 2020b), we call the multi-step method trajectory fitting as we use the solver to construct a trajectory and compare it with the ground truth obtained through a high-precision solver; a combination of small time steps and a higher-order solver. The loss is MSE for both position and velocity across the entire trajectory where $\hat{q}_n, \hat{v}_n$ and q_n, v_n denote the position and velocity of our method and the ground truth respectively for time step n. To construct equivalent trajectories, we let $\hat{q}_0, \hat{v}_0 = q_0, v_0$. For a trajectory with horizon N, the loss is computed as follows where $\|\cdot\|_2$ denotes an ℓ^2-norm

$$\mathcal{L}_{traj} = \frac{1}{N}\sum_{n=1}^{N} \|\hat{q}_n - q_n\|_2^2 + \|\hat{v}_n - v_n\|_2^2 \tag{4.5}$$

We can also utilize the alternative training method, single step training, which we modify to predict the scaled local truncation error, called the residual.

$$\mathcal{R}(q_n, v_n, \tilde{q}_n, \tilde{v}_n, h) = \frac{1}{h^3}\begin{bmatrix} q_n - \tilde{q}_n \\ v_n - \tilde{v}_n \end{bmatrix} \tag{4.6}$$

where $\tilde{q}_n, \tilde{v}_n = VV(q_{n-1}, v_{n-1})$ is computed by velocity Verlet. The training method is called residual training by (Shen et al., 2020) and (Poli et al., 2020b). The variables q_n, v_n denote the position and velocity of the ground truth, and similarly to trajectory fitting, we let $\hat{q}_0, \hat{v}_0 = q_0, v_0$. The loss function is MSE of the residual rather than the position and velocity.

$$\mathcal{L}_{res} = \frac{1}{N}\sum_{n=1}^{N} \|g_\theta(v_{n-1}, a(q_{n-1}), \tilde{v}_n, a(\tilde{q}_n), m, \Gamma) - \mathcal{R}(q_n, v_n, \tilde{q}_n, \tilde{v}_n, h)\|_2^2 \tag{4.7}$$

For certain systems, physical priors can be incorporated into the loss function as regularization to improve the conversation of these quantities. E.g. for MD systems, it is common to shift the center of mass to $(0, 0)$, and if we assume objects indexed by i, we can compute the center of mass as $\sum_i (q_i \cdot m_i) / \sum_i m_i$. The regularization can then be the mean square of center of mass. Similarly, if there is a conservation of total momentum $\sum_i p_i$, this can be incorporated into the loss function as regularization.

4.3 Theoretical analysis

While empirical results may suffice to show improvements compared to traditional solvers, proofs of the properties such as the local truncation error yield insight into why HyperVerlet may outperform a traditional solver, and show that HyperVerlet is symplectic if the neural network is a symplectomorphism.

Starting with the local truncation error, we follow the proof technique of both (Shen et al., 2020) and (Poli et al., 2020b) where the core assumption is that the neural network corrector is a $O(\eta)$ approximator of the residual $\mathcal{R}$.

Theorem 1 *If g_θ is a $O(\eta)$ approximator of $\mathcal{R}$ then HyperVerlet has a local trunction error of $O(\eta h^3)$.*

Proof Since g_θ is a $O(\eta)$ approximator of $\mathcal{R}$, we have for all $1 \leq n \leq N$ that

$$\|\mathcal{R}(q_n, v_n, \tilde{q}_n, \tilde{v}_n, h) - g_\theta(v_{n-1}, a(q_{n-1}), \tilde{v}_n, a(\tilde{q}_n), m, \Gamma)\| \leq O(\eta) \tag{4.8}$$

For a HyperVerlet, by definition the local truncation error is

$$\begin{aligned}
\tau_n &= \left\| \begin{bmatrix} q_n \\ v_n \end{bmatrix} - \begin{bmatrix} \hat{q}_n \\ \hat{v}_n \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} q_n \\ v_n \end{bmatrix} - \begin{bmatrix} \tilde{q}_n \\ \tilde{v}_n \end{bmatrix} - h^3 \cdot g_\theta(v_{n-1}, a(q_{n-1}), \tilde{v}_n, a(\tilde{q}_n), m, \Gamma) \right\| \\
&= \left\| h^3 \cdot \mathcal{R}(q_n, v_n, \tilde{q}_n, \tilde{v}_n, h) - h^3 \cdot g_\theta(v_{n-1}, a(q_{n-1}), \tilde{v}_n, a(\tilde{q}_n), m, \Gamma) \right\| \\
&\leq O(\eta h^3)
\end{aligned} \tag{4.9}$$

■

Since the local truncation error of the HyperVerlet solver is $O(\eta h^3)$, the order of the solver remains 2, but a factor η better than velocity Verlet if $\eta < 1$. To prove that the global truncation is $O(\eta h^2)$ and the solver is convergent, it is sufficient to show that the HyperVerlet solver is Lipschitz continuous in $z = (q, v)$ since it is a single-step method (Süli, 2003). Note that the class of fully-connected neural networks with a Lipschitz continuous activation function, e.g. sigmoid and ReLU families, is Lipschitz continuous (Xu & Mannor, 2011). Similarly, a large class of graph neural networks are also Lipschitz continuous (Dasoulas et al., 2021).

To derive that the solver is Lipschitz continuous in z, we start by proving a useful axiom for Lipschitz continuity wrt. any variable x.

Lemma 2 *For an arbitrary function f that can be decompose into the sum of two other functions $f(x, t) = g(x, t) + h(x, t)$ is Lipschitz continuous in x if both of the two functions g and h are Lipschitz continuous in x.*

Proof For an arbitrary function $f(x,t) = g(x,t) + h(x,t)$, assume both g and h are Lipschitz continuous. By the definition of Lipschitz continuity, we have that

$$\|g(x(t_1),t_1) - g(x(t_2),t_2)\| \leq K_g\|x(t_1) - x(t_2)\| \tag{4.10}$$
$$\|h(x(t_1),t_1) - h(x(t_2),t_2)\| \leq K_h\|x(t_1) - x(t_2)\| \tag{4.11}$$

By the triangle inequality of a norm, that is $\|a+b\| \leq \|a\| + \|b\|$, the following statement with $K = K_g + K_h$ is true

$$\|f(x(t_1),t_1) - f(x(t_2),t_2)\| \leq K\|x(t_1) - x(t_2)\| \tag{4.12}$$

and thus, f is Lipschitz continuous in x. ■

Now, since HyperVerlet is the sum of the velocity Verlet predictor and the neural network corrector, which we know is Lipschitz continuous for a large class of neural networks, we want to show Lipschitz continuity for the velocity Verlet.

Lemma 3 *Velocity Verlet is Lipschitz continuous in $z = (q,v)$.*

Proof By our definition of an IVP, see Definition 7, and Hamilton's equations, the acceleration function a is Lipschitz continuous in q.

$$\frac{dz}{dt} = \begin{bmatrix} \frac{\partial H(q,p)}{\partial p} \\ -\frac{1}{m} \cdot \frac{\partial H(q,p)}{\partial q} \end{bmatrix} = \begin{bmatrix} v(t) \\ -a(q(t)) \end{bmatrix} \tag{4.13}$$

where $z_0 = (q_0, v_0)$ are the initial conditions and m is the mass.

Now, recall that a velocity Verlet is computed by

$$q_{n+1} = q_n + h \cdot v_n + \frac{h^2}{2}a(q_n) \tag{4.14}$$
$$v_{n+1} = v_n + h \cdot \frac{a(q_n) + a(q_{n+1})}{2} \tag{4.15}$$

This update step can be represented with the following functional abstraction where VV should be Lipschitz continuous in z.

$$z_{n+1} = VV(z_n, h) \tag{4.16}$$

Starting with Equation 4.14, it is trivial to show that

$$\|q_a - q_b\| \leq K_q\|q_a - q_b\| \tag{4.17}$$
$$\|h \cdot (v_a - v_b)\| \leq K_v\|v_a - v_b\| \tag{4.18}$$

for some $K_q, K_v \in \mathbb{R}^+$, and thus, these two terms satisfy the Lipschitz condition. By definition a is Lipschitz continuous in q and using Lemma 2, Equation 4.14 is Lipschitz continuous in z. A similar argument applies to Equation 4.15 with v_n and a. Therefore, we can conclude velocity Verlet is Lipschitz continuous in z. ■

With these properties of Lipschitz continuity and velocity Verlet, we can establish that HyperVerlet is Lipschitz continuous.

Theorem 4 *If g_θ is Lipschitz continuous in z then HyperVerlet is Lipschitz continuous in z.*

Proof Since velocity Verlet is Lipschitz continuous in z and by the conditions of the theorem, g_θ is Lipschitz continuous in z, then by Lemma 2, HyperVerlet is Lipschitz continuous in z. ■

Due to Theorem 4, we can conclude HyperVerlet has a global truncation error of $O(\eta h^2)$ and is convergent.

The final desirable property, symplecticity, is unfortunately not achieved by HyperVerlet since the corrector is not guaranteed to be symplectic. Recent efforts (Jin et al., 2020) have shown that it is possible to design a neural network that results in symplectic transformations, which may have prospect for reinstating this property for our method. The problem of this method is that it blocks generalization across families of phase spaces, but we have a theoretical solution to the problem. See Appendix C.1 for more details.

Chapter 5

Experiment systems

To analyze the performance of our novel method, HyperVerlet, we will conduct experiments on a wide variety of Hamiltonian systems including single-body, multi-body, linear, non-linear, non-chaotic, and chaotic systems. In this chapter, we will define the systems by their Hamiltonian equations.

5.1 Undamped spring-mass system

Perhaps the most widely studied system is the idealized spring-mass system where a spring is mounted on a fixed surface at one end and have a moving bob described by a point-mass m at the other end, see Figure 5.1. The spring has an idle position l such that if the spring is stretched beyond this point, it will attract the bob, and if it is squeezed, it will repel the bob. We denote the position of the bob q, and thus the displacement from the idle position is $q - l$. From a Newtonian perspective, the force applied is inversely linear to the displacement from the idle position by a constant k called the spring constant. This relations is called Hooke's law.

$$F = -k \cdot (q - l) \tag{5.1}$$

We can derive this equation of motion, or force, from the energy of the system where the potential energy V is given by $\frac{1}{2}k(q-l)^2$ and the kinetic energy $T = \frac{1}{2}mv^2$. Using Hamilton's equations, in addition to the force equation, we also derive the equation for velocity.

$$F = -\frac{\partial H}{\partial q} = -\frac{\partial}{\partial q}\left(\frac{1}{2}m\dot{q}^2 + \frac{1}{2}k(q-l)^2\right) = -k(q-l) \tag{5.2}$$

$$v = \frac{\partial H}{\partial p} = \frac{\partial}{\partial p}\left(\frac{1}{2}mv^2 + \frac{1}{2}k(q-l)^2\right) = \frac{\partial}{\partial p}\frac{p^2}{2m} = \frac{p}{m} \tag{5.3}$$

The motion of the spring-mass is called a linear oscillator and is analytically solvable and non-chaotic.

$$q(t) = \frac{v_0}{\omega}\sin(\omega t) + (q_0 - l)\cos(\omega t) \tag{5.4}$$

where $\omega = \sqrt{k/m}$. The Hookian spring model is also the origin of the term *stiff systems* meaning dynamical systems that are difficult to solve with discrete integrators. The name is derived from

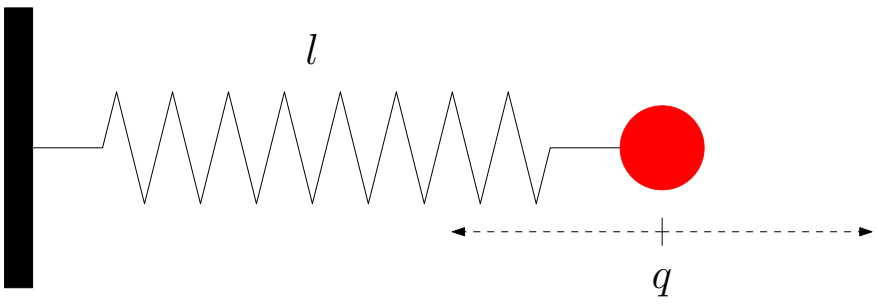


Figure 5.1: Idealized spring-mass system where the bob (red) is approximated by point mass and the mass of the spring is omitted. There are no frictional forces, and thus, the bob should travel back and forth infinitely if $q_0 \neq l$ and $v_0 \neq 0$.

the fact that for a large spring constant k, the system changes rapidly, which makes it difficult to solve without extremely small time steps.

For this system, the phase space is 2-dimensional, and with the derivative of the phase space before and after the velocity Verlet Update, g_θ should accept a 4-dimensional input plus the mass m, the spring constant k, and the idle length l, summing to a vector input of size 7. We employ a feed forward neural network with 3 hidden layers and sigmoid activation for this experiment system.

5.2 Ideal pendulum

Another oscillating, deceptively simple, Hamiltonian system is the pendulum with two primary distinguishing characteristic: polar coordinates and a non-linear equation of motion. The polar coordinates are a natural choice as can be seen on Figure 5.2 since it allows treating one coordinate, the length of the pendulum, as a constant. To see the equation of motion is non-linear observe that the potential energy of the gravitational force acting upon the bob is

$$V = mg \cdot (q_r - d) = mgq_r \cdot (1 - \cos q_\alpha) \tag{5.5}$$

where m is the mass, g is the gravitational acceleration (approximately $9.87 m/s^2$ on Earth), q_r is the length of the pendulum, and d is the vertical distance from the origin plane to the bob. By applying Hamilton's equations, we can derive the equations of motion. Note that since the coordinate system is non-linear, we need to compute the momentum from the Lagrangian

$$p_r = \frac{\partial L}{\partial \dot{q}_r} = \frac{\partial}{\partial \dot{q}_r}\left(\frac{1}{2} m q_r^2 \dot{q}_\alpha^2 + mgq_r \cdot (1 - \cos q_\alpha)\right) = 0 \tag{5.6}$$

$$p_\alpha = \frac{\partial L}{\partial \dot{q}_\alpha} = \frac{\partial}{\partial \dot{q}_\alpha}\left(\frac{1}{2} m q_r^2 \dot{q}_\alpha^2 + mgq_r \cdot (1 - \cos q_\alpha)\right) = m q_r^2 \dot{q}_\alpha \tag{5.7}$$

As we can see from Equation 5.6, the momentum of the length is zero, coherent with the constant length of the pendulum. Therefore, the force along this direction remains zero too. The

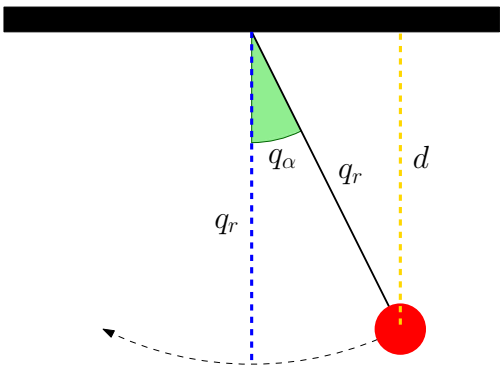


Figure 5.2: Non-frictional pendulum where the bob (red) is approximated by point mass. The system is undamped, and thus, the bob should swing back and forth infinitely if $q_0 \neq 0$ and $v_0 \neq 0$.

interesting coordinate is q_α with its associated force and velocity.

$$\dot{q}_\alpha = \frac{\partial H}{\partial p_\alpha} = \frac{\partial}{\partial p_\alpha}\left(\frac{1}{2}mq_r^2\dot{q}_\alpha^2 + mgq_r \cdot (1-\cos q_\alpha)\right) = \frac{p_\alpha}{mq_r^2} \tag{5.8}$$

$$F_\alpha = -\frac{\partial H}{\partial q_\alpha} = -\frac{\partial}{\partial q_\alpha}\left(\frac{1}{2}mq_r^2\dot{q}_\alpha^2 + mgq_r \cdot (1-\cos q_\alpha)\right) = -mgq_r \sin q_\alpha \tag{5.9}$$

In contrast to the spring-mass system, there exists no analytical solution to the differential equation describing this system. For small angles, there exists an approximate solution because $\sin q_\alpha \approx q_\alpha$ for $q_\alpha \ll 1$, but for larger angles, we must rely on numerical approximations.

Similarly to the spring-mass system, we employ a feed-forward neural network with 4-dimensional vector input for the phase space derivatives. Note that the configuration space of the pendulum is 2-dimensional as opposed to the 1-dimensional configuration space of spring-mass, but since the magnitude, i.e. length of the pendulum, is irrelevant as the momentum in this direction is always zero, we only model the angle. The task specific parameters for the pendulum system are the pendulum length l and gravitational constant g.

5.3 Molecular dynamics

Due to the regularity of molecular dynamics simulations, we can define it as an entire family of simulation with little variability (Vollmayr-Lee, 2020). A common feature is the Cartesian space in two or three dimensions, which implies that the relationship between momentum and velocity is $p = mv$ invariant to the dynamics of the system. Simulations with other coordinate systems do exist but the majority is Cartesian, and we will only focus on this type.

To progress the simulation through time, we need to derive the equations of motion, which for the velocity is simply $v = p/m$. We compute the forces as the negative derivative of the Hamiltonian wrt. position, and since only the potential energy is a function of position, it is the negative derivative of the potential energy wrt. position, i.e. $\dot{p}_i = -\frac{\partial V(q)}{\partial q_i}$. As a result, it is only necessary to characterize the potential energy of the system to vary the dynamics. The family of molecular dynamics simulations we focus on is characterized by a two-body or pair potential meaning that the energy is derived from the distance between all pairs of particles, that is

$$V(q) = \frac{1}{2}\sum_{i\neq j} V(q_i, q_j) \tag{5.10}$$

where V is the symmetric pair potential function. Notice the multiplication by one half to ensure that the energy of a binding is counted only once. With this definition, we can refine the computation of the net forces acting upon any particle i as the sum of forces from each binding with other particles j.

$$F_i = -\frac{\partial V}{\partial q_i} = -\frac{1}{2}\sum_{j\neq i}\frac{\partial V(q_i, q_j)}{\partial q_i} \tag{5.11}$$

For faster computations, it is common to leverage that the potential energy between two particles converges to zero as the distance between the particles increases by defining a cut-off c where the potential energy between two particles is zero, $V(q_i, q_j) = 0$, if the distance is greater than this threshold, $\|q_i - q_j\| > c$.

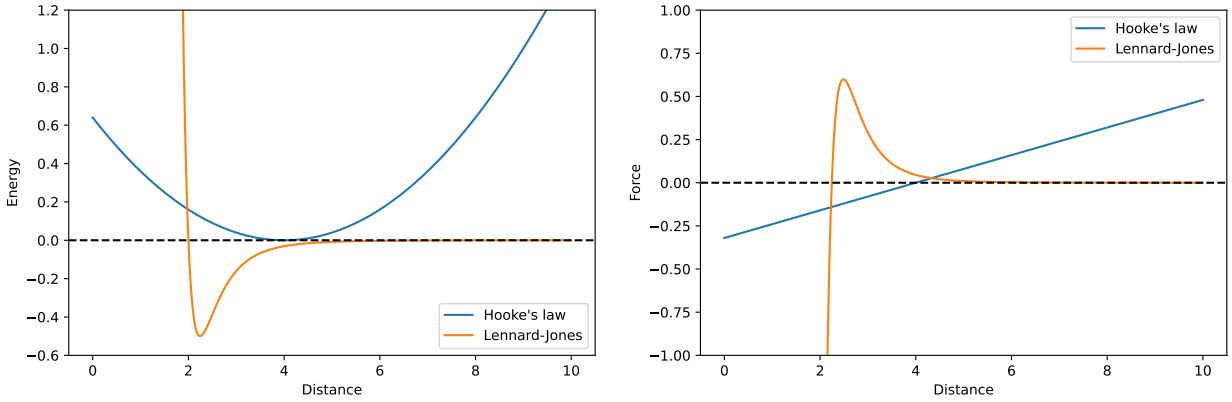


Figure 5.3: Two potential energy functions and their associated force functions. For Hooke's law, the idle length l is 4 and spring constant k is 0.08. For Lennard-Jones, the zero-energy distance σ is 2, and the well-depth ε is 0.5.

The force computation for a pair potential is sufficient for the correctness of a molecular dynamics simulation, but for realistic simulations, the required number of atoms to simulate is infeasible with current technology. To reduce the number required, one can define an infinite molecular space with finitely many atoms through the use of periodic boundary conditions. Periodic boundary conditions is similar to the arcade game Asteroids where objects leaving one side of the space loop back to the exact opposite side along each axis retaining its momentum. Practically, a particle $q = (x, y, z)$ in a simulation box $L = (L_x, L_y, L_z)$ has infinitely many positions with periodicity equivalent with the simulation box length.

$$q = (x + n_x L_x, y + n_y L_y, y + n_y L_y) \quad \text{for all } n_x, n_y, n_z \in \mathbb{Z} \tag{5.12}$$

Since there are now infinitely many distances between any pair of particles, we must rely on the minimum image convention where only the closest position, or image, of a particle relative to another particle is considered.

For molecular dynamics, multiple types of simulations exists, and the simulation derived above is called NVE for constant number of particles, volume, and energy. As a result, the dynamics of simulation type can be described by Hamilton's equations. Other types of simulations include NVT and NPT where the number of atoms and temperature is constant for both types, and volume and pressure is constant respectively. The experiments are only conducted on NVE simulations for simpler dynamics, but we conjecture that the method generalizes to other types of simulations too.

By the translational invariance of Hamiltonian systems, the drift of the system does not impact the dynamics, but avoiding it is convenient for tasks like plotting and comparison between different systems. To avoid the drift, we can initialize the total momentum to zero and rely on the conservation of momentum because the momentum dictates the velocity and by extension the drift of the system. To this end, assuming the velocity of each particle v_i is drawn from a probability distribution $v_i \sim p(v)$ and the mass is constant, a shift in initial velocity is sufficient to achieve zero total momentum.

$$\tilde{v}_i = v_i - \frac{1}{N} \sum_j v_j \tag{5.13}$$

where N is the number of particles. Similarly, we can center the system around the origin $(0, 0)$ by subtracting the center of mass, $\bar{q} = (\sum_i q_i \cdot m_i) \,/\, (\sum_i m_i)$, from the position of all particles.

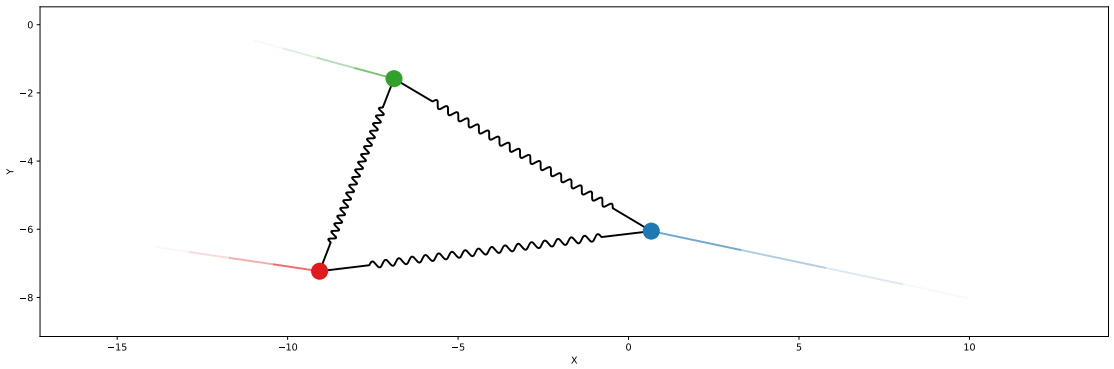


Figure 5.4: Example of a spring potential molecular dynamics system with three particles in a 2D-plane. The springs have different idle lengths and spring constants, and the particles have different masses. The plot includes a trail for each particle to observe the latest part of the trajectory.

The neural network corrector g_θ for this class of systems is much different from the spring-mass and pendulum systems. Here, we employ a GN because we want to apply the same error correction dynamics to all particles, each represented by a node in the graph. Additionally, the scalability property of GNs allows changing the size of the simulation domain, i.e. number of particles, while applying the same corrector.

With this framework for molecular dynamics, we can define individual experiments by their potential energy and force function, see Figure 5.3. We conduct our experiments with the spring potential as described below, but by replacing the potential function, HyperVerlet will work for other MD simulations. One such example is the Lennard-Jones potential, a non-linear two-term potential, and a description of this can be found in Appendix A.

5.3.1 Spring potential

Similarly to the undamped spring-mass system, we can let molecular dynamics simulations be governed by Hooke's law. Between all pairs of particles, there is suspended a zero-mass spring with a spring constant k. The equation governing the force acting upon each particle i is

$$\begin{aligned} F_i &= -\frac{1}{2}\sum_{i\neq j}\nabla_{q_i}V(q_i,q_j) \\ &= -\frac{1}{2}\sum_{i\neq j}\nabla_{q_i}\frac{1}{2}k_{ij}(\|q_j-q_i\|-l_{ij})^2 \\ &= -\frac{1}{2}\sum_{i\neq j}k_{ij}(\|q_j-q_i\|-l_{ij})\frac{q_i-q_j}{\|q_j-q_i\|} \end{aligned} \tag{5.14}$$

where the spring constant $k_{ij} = k_{ji}$ and the spring length $l_{ij} = l_{ji}$ are symmetric.

The specific experiment setup we employ for this system is 2-dimensional and consists of three particles with slightly different masses, and similarly slightly different spring constants and lengths drawn from a Gaussian distribution. An example of this setup can be observed in Figure 5.4. Even with this few particles, the system is chaotic, which makes an ideal study of the efficacy of Hamiltonian hypersolvers.

Chapter 6

Results

To empirically investigate the performance of the HyperVerlet, we test it on three physical tasks of increasing difficulty. The first and easiest being the spring mass system as there is an analytical solution to the system. The second and harder system is the undamped pendulum, as there is no exact analytical solution and is represented by polar coordinates. The third and last system is the three body problem, a chaotic system where small errors in the prediction are penalized hard, making it the hardest system to predict.

Since MSE only measures accuracy over the entire trajectory not accounting for instability and penalizes accumulating errors significantly, we also choose to compute the Valid Prediction Time (VPT), as defined in (Vlachas et al., 2020) and used in (Jin et al., 2020). VPT is defined as:

$$\text{VPT}_\varepsilon = \arg\max\{t_f | RMSE(\hat{z}(t)) < \varepsilon, \forall t \leq t_f\} \tag{6.1}$$

This measure is an interpretation of how long the prediction remains valid. For the experiments made in this thesis, we choose an ϵ of value 0.1. Source code and videos of the simulations can be found at `https://github.com/Zinoex/hyperverlet`.

6.1 Integrator comparison

To test the performance of our proposed method, the HyperVerlet, we compare it to the performance of different solvers by creating 100 high-precision ground truth simulations made with a FR4 solver using a finer step size. The high resolution trajectory of the ground truth simulation is then coarsened by an integer factor, resulting in high step size. The solvers are then tested on their ability to reproduce the same trajectory as the ground truth simulation, but with a higher step size as well as their reliability measured by VPT.

As the main metric to compare results, we use MSE between ground truth and the predicted canonical coordinates averaged across the time axis and the phase space for the entire trajectory and averaged over the 100 different test configurations, but we find that VPT yields better insight into the stability of the prediction.

6.1.1 Spring-mass

For the spring-mass system, the high-precision simulation was created with a duration of 1000 time units composed of a total of 1000000 samples, making the step size of the ground truth simulation equivalent to $\frac{1000}{1000000} = 0.001$. The solvers were tested on the spring-mass system for four different step sizes $[0.025, 0.05, 0.1, 0.2]$.

From the results presented in Table 6.1, it is evident that the choice of solver impact the accuracy of the simulation. It is clear to see how Euler method got its reputation as a poor solver as it, even for smaller step size, achieves an MSE score many magnitudes higher compared to the other solvers. For the HyperEuler, we observe that as the step size reaches $h = 0.1$ HyperEuler becomes unstable in some of the test configurations, leading to its poor performance.

Common for all of the hypersolvers in Table 6.1 is how they outperform their traditional base solver by orders of magnitude. It is unsurprising that HyperHeun outperforms HyperEuler, since it is a 2^{nd}-order solver. Comparing the accuracy of HyperHeun and HyperVerlet, we see

Table 6.1: MSE of different solvers tested over different step sizes on the spring-mass system. Smaller is better.

Solver	$h = 0.025$	$h = 0.05$	$h = 0.1$	$h = 0.2$
Euler	2.272e+07	4.866e+16	∞	∞
HyperEuler	2.580e−04	3.839e−02	3.448e+17	∞
Heun	2.885e−04	4.622e−03	7.198e−02	4.471e−01
HyperHeun	2.046e−08	1.762e−07	2.685e−06	4.349e−05
Velocity Verlet	1.809e−05	2.883e−04	4.587e−03	6.722e−02
HyperVerlet	9.676e−09	3.912e−08	5.275e−07	8.882e−06
FR4	7.168e−09	1.143e−08	9.385e−07	2.377e−04
RK4	7.334e−09	7.131e−09	1.804e−08	3.667e−06

Table 6.2: VPT of different solvers tested over different step sizes on the spring-mass system measured. Higher is better.

Solver	$h = 0.025$	$h = 0.05$	$h = 0.1$	$h = 0.2$
Euler	32.9	16.7	8.4	4.2
HyperEuler	997.3	953.9	887.3	659.4
Heun	1000.0	800.7	298.6	93.2
HyperHeun	1000.0	1000.0	1000.0	1000.0
Velocity Verlet	1000.0	1000.0	802.2	302.3
HyperVerlet	1000.0	1000.0	1000.0	1000.0
FR4	1000.0	1000.0	1000.0	1000.0
RK4	1000.0	1000.0	1000.0	1000.0

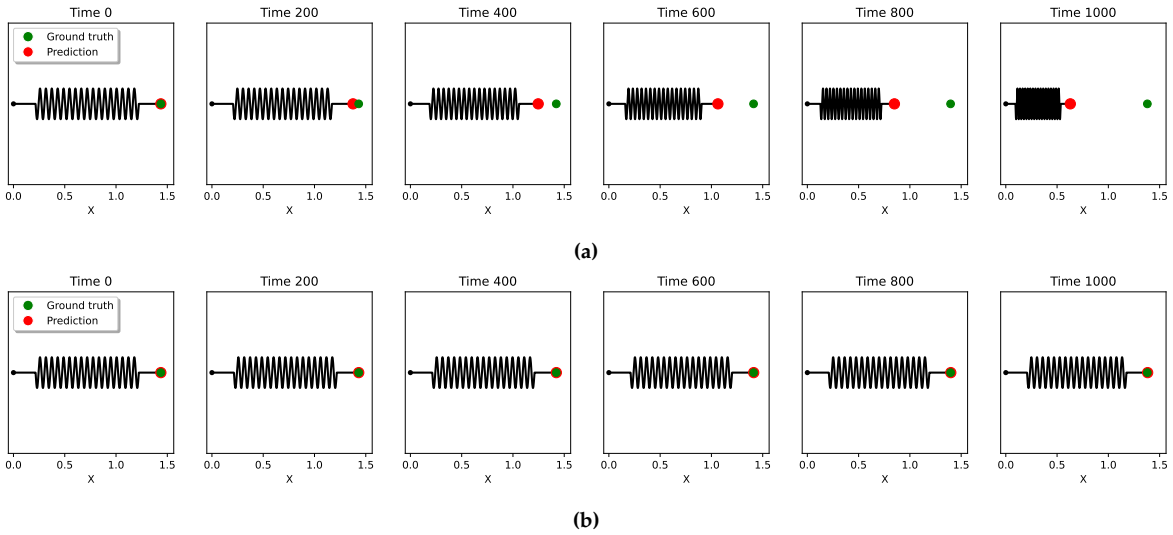


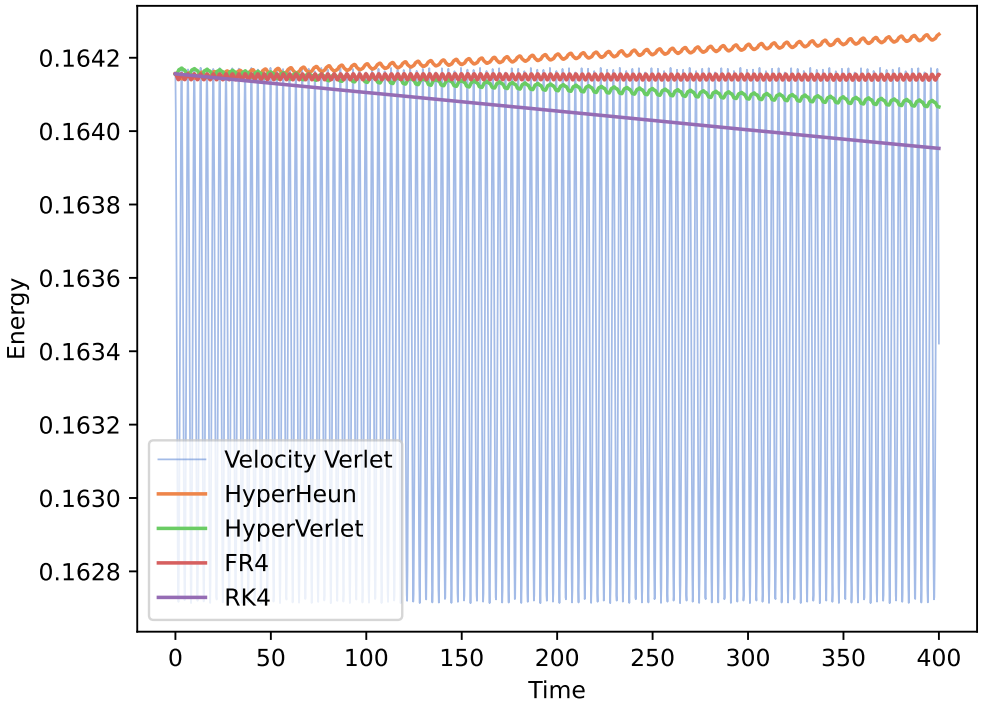
Figure 6.1: 6 equally spaced snapshots of a simulation rollout of the spring mass system. **(a)** shows velocity Verlet and **(b)** shows HyperVerlet. Both systems are computed using a step size of $h = 0.2$ compared to the ground truth with a step size of $h = 0.001$. The prediction has visually been made slightly larger then the ground truth to make is visible during overlap.

that HyperVerlet outperforms HyperHeun by almost an order of magnitude regardless of the step size, although both methods are of 2^{nd}-order, showing the power of a symplectic base solver.

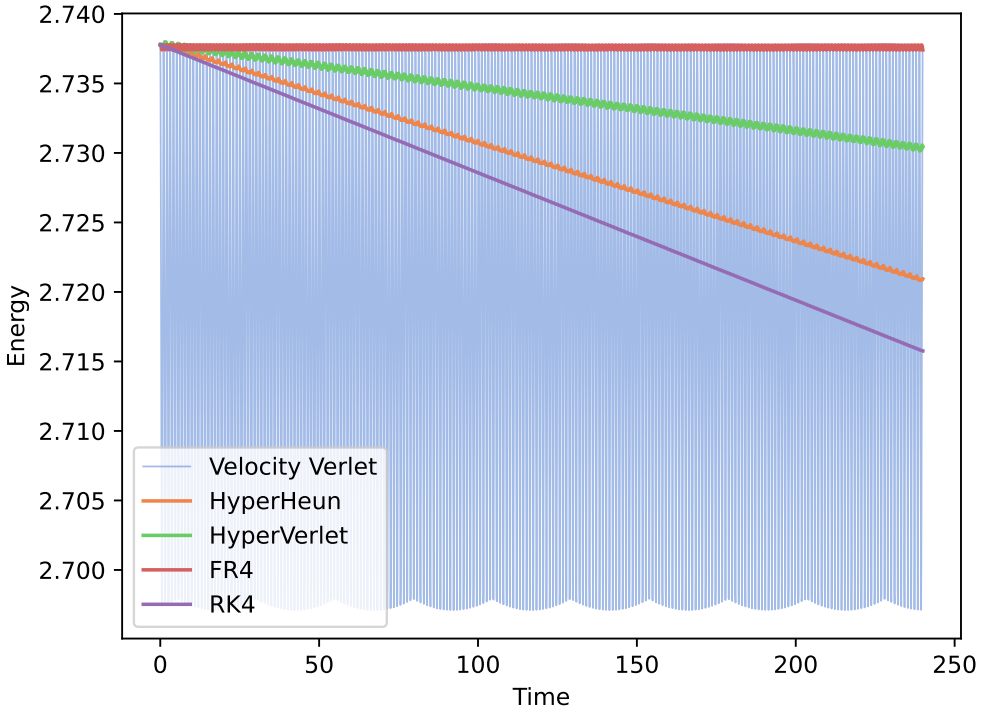
Comparing the 2^{nd}-order HyperVerlet to the two 4^{th}-order solvers FR4 and RK4, we can see that for a low step size, $h = 0.025$, HyperVerlet is almost on par with both 4^{th}-order solvers. For large step sizes, $h = 0.2$, the performance of HyperVerlet is two orders of magnitude better than FR4 and almost on par with RK4.

One of the downsides of using a symplectic solver like the velocity Verlet is that in order to conserve the total energy of the system, the trajectory of bob tends to desynchronize with the ground truth trajectory as the step size increases. This occurs despite the total energy of the system oscillates around the true energy, hence velocity Verlet trades conservation of energy for desynchronization of the bob as can be seen in Figure 6.2a. An example of this issue for a single configuration is shown in Figure 6.1 where it is evident from Figure 6.1a that as time progresses, the prediction by velocity Verlet desynchronizes with the ground truth. This, however, is not the case for HyperVerlet where Figure 6.1b shows how the prediction stays in sync with the ground truth. This result is also evident from Table 6.2 where we see that as the step size increases, the VPT of velocity Verlet starts to decrease, leaving it with a VPT of 302.3 for a step size of $h = 0.2$, whereas HyperVerlet achieves a valid prediction throughout the whole duration.

By examining Figure 6.2a, we see how the total energy changes over the first 40% duration of the simulation. The figure shows how the total energy for velocity Verlet oscillates within a fixed bound. Comparing the amplitude of the energy oscillation and the total energy for HyperVerlet, we see that although HyperVerlet loses energy, less than the higher order solver RK4, it would require very long durations before the total energy becomes less than the amplitude of the oscillation for velocity Verlet. Note that the trends continue for the full trajectory but we only show the start to highlight the oscillating behavior of velocity Verlet. Additionally, we exclude Euler, HyperEuler, and Heun because of the amount of energy they accumulate or dissipate. A figure of the total energy for the full duration with all solvers can be found in Appendix B.1a.



(a)



(b)

Figure 6.2: Both figures show the total energy of a single trajectory for a system for the first 40% of the rollout comparing the different solvers. Euler, HyperEuler and Heun are excluded for clarity. A plot with these over the full trajectory can be found in Appendix B. **(a)** shows the total energy for the spring-mass system. **(b)** shows the total energy for the pendulum system.

Table 6.3: MSE of different solvers tested over different step sizes on the ideal pendulum system. Smaller is better.

Solver	$h = 0.02$	$h = 0.04$	$h = 0.06$	$h = 0.08$
Euler	2.103e+07	4.117e+07	5.267e+07	5.182e+07
HyperEuler	2.210e+00	2.484e+00	2.558e+00	2.626e+00
Heun	2.139e+00	3.261e+00	9.192e+03	1.026e+06
HyperHeun	1.221e+00	7.016e−02	7.225e−01	5.703e−01
Velocity Verlet	2.819e−02	4.284e−01	1.794e+00	3.453e+00
HyperVerlet	5.200e−03	3.905e−02	1.357e−01	1.005e−01
FR4	1.283e−05	5.816e−05	1.093e−03	1.063e−02
RK4	1.649e−05	8.793e−04	4.756e−02	7.026e−01

Table 6.4: VPT of different solvers tested over different step sizes on the ideal pendulum system measured. Higher is better.

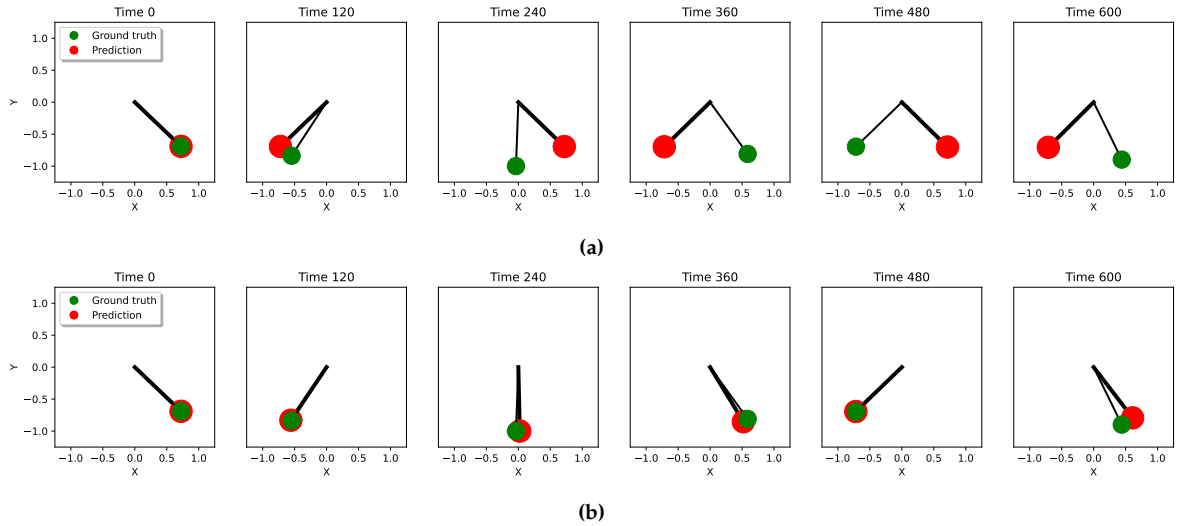
Solver	$h = 0.02$	$h = 0.04$	$h = 0.06$	$h = 0.08$
Euler	2.1	1.1	0.8	0.6
HyperEuler	236.9	172.3	177.3	186.1
Heun	143.6	49.4	26.2	13.5
HyperHeun	305.3	418.9	331.0	280.1
Velocity Verlet	252.3	104.5	58.8	41.1
HyperVerlet	510.4	451.3	450.6	338.2
FR4	600.0	600.0	581.9	376.2
RK4	600.0	577.4	404.9	303.4

6.1.2 Pendulum

For the pendulum system, the high-precision simulation was created with a duration of 600 time units composed of a total of 600000 samples, making the step size of the ground truth simulation equivalent to $\frac{600}{600000} = 0.001$. The solvers were tested on the pendulum system for four different step sizes $[0.02, 0.04, 0.06, 0, 08]$.

By inspecting the results in Table 6.3 and 6.4, we see many of the same conclusions as were made for the spring-mass system in Section 6.1.1. We see Euler is a bad choice of solver, that hypersolvers overall outperform their traditional non-hypersolver, and how HyperVerlet outperforms other hypersolvers. One interesting thing to note in Table 6.3 and 6.4 is how there is a large difference in accuracy comparing symplectic and non-symplectic solvers, even for higher order solvers. E.g. as the step size increases the accuracy of FR4 declines as expected. However, it is much less than the rate by which the accuracy of RK4 decreases, showing the importance of symplecticity in longer simulations, which allows the solver to conserve the energy of the system throughout the simulation. Comparing RK4 and HyperVerlet, we see that RK4 is better for smaller step sizes, but degrades faster, which we attribute to the symplecticity of the base solver of HyperVerlet.

Although symplecticity is important for longer durations, our experiments show a disadvantage of having this property. As mentioned in Section 6.1.1, the prediction tends to deviate from the ground truth sooner as the step size increases, which penalizes the VPT measure as is also



(a)

(b)

Figure 6.3: 6 equally spaced snapshots of a simulation rollout of the pendulum system. **(a)** shows velocity Verlet and **(b)** shows HyperVerlet. Both systems are computed using a step size of $h = 0.08$. The prediction bob has been made visually larger than the ground truth to make it visible during overlap.

evident from Table 6.4. This desynchronization behavior can be seen in Figure 6.3, showing how velocity Verlet in Figure 6.3a desynchronizes with the ground truth already at 200 time units, whereas HyperVerlet shown in Figure 6.3b remains synchronized for more than 400 time units.

By inspecting the total energy in the system seen on Figure 6.2b, we see many of the same trends as in Figure 6.2a. It is evident that HyperVerlet loses less energy than both RK4 and HyperHeun and remains within the amplitude of velocity Verlet for a very long period.

6.1.3 Three-body spring-mass

For the three-body spring-mass system, the high-precision simulation is created with a lower time step compared to the spring-mass and pendulum systems due to the chaotic nature of the system. The duration of the trajectory is 30 time units composed of a total of 100000 samples, making the step size of the ground truth simulation equivalent to $\frac{30}{100000} = 0.0003$. The solvers were tested on the three-body spring-mass system for four different step sizes $[0.0075, 0.015, 0.03, 0.06]$.

From the results shown in Table 6.5, comparing the different solver on the three-body spring-system, we see that HyperVerlet for all step sizes outperform the velocity Verlet as well as both hypersolvers. From Table 6.5, we see how all 2^{nd}-order solvers achieve results within the same order of magnitude, and the same result can be seen for both of the 4^{th}-order solvers. We can derive the same conclusions when measuring VPT shown in Table 6.6. We contribute this to the chaotic nature of the three-body spring-system, as tiny errors along the trajectory completely change the course of the simulation, along with the system being very sensitive to the initial condition of the system. Therefore, we hypothesize that chaotic systems like the three-body spring-mass system may pose a great challenge for machine learning algorithms as they only approximate the dynamics. This is also evident from Figure 6.4a where we see that velocity Verlet at time 12 provides a stable prediction, but in the next snapshot at time 18, we see how the prediction has deviated from the ground truth. The same conclusion is true for HyperVerlet

Table 6.5: MSE of different solvers tested over different step sizes on the three-body spring-mass system. Smaller is better.

Solver	$h = 0.0075$	$h = 0.015$	$h = 0.03$	$h = 0.06$
Euler	4.235e+00	9.818e+00	2.644e+01	9.536e+01
HyperEuler	1.313e+00	1.243e+00	1.931e+00	5.957e+00
Heun	9.152e−02	2.821e−01	4.891e−01	8.301e−01
HyperHeun	8.370e−02	1.922e−01	4.653e−01	8.626e−01
Velocity Verlet	7.977e−02	2.601e−01	4.618e−01	8.400e−01
HyperVerlet	4.794e−02	1.847e−01	4.037e−01	7.550e−01
FR4	4.097e−02	4.887e−02	1.476e−01	4.565e−01
RK4	2.313e−02	5.609e−02	1.545e−01	1.635e−01

Table 6.6: VPT of different solvers tested over different step sizes on the three-body spring-mass system measured. Higher is better.

Solver	$h = 0.0075$	$h = 0.015$	$h = 0.03$	$h = 0.06$
Euler	5.7	3.5	2.0	1.0
HyperEuler	11.9	11.3	9.0	6.6
Heun	28.4	26.8	24.1	14.4
HyperHeun	28.8	27.0	24.8	20.4
Velocity Verlet	28.5	26.9	24.7	21.0
HyperVerlet	28.8	26.9	24.9	21.3
FR4	28.9	28.1	27.1	24.6
RK4	29.2	28.6	27.9	26.7

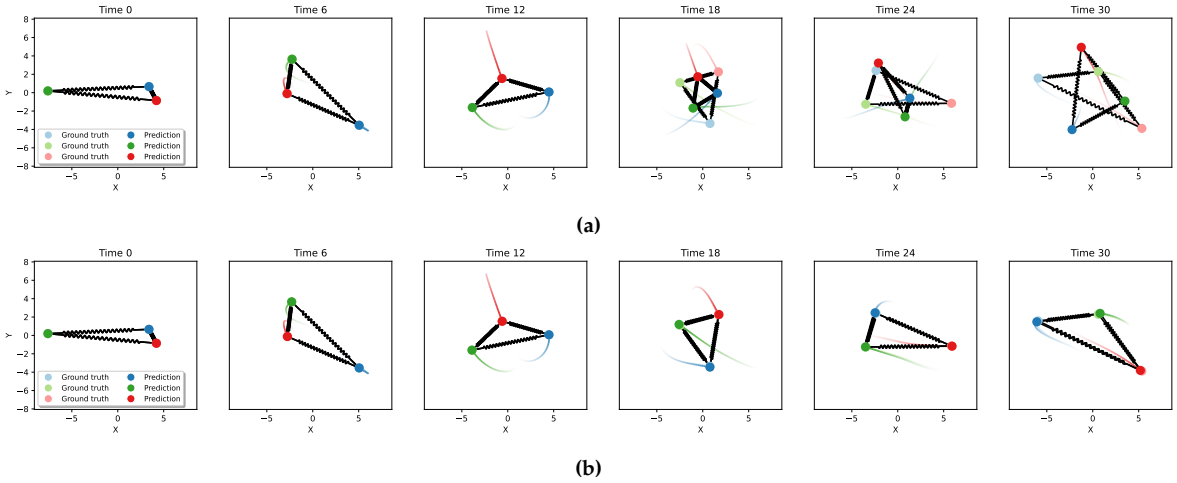


Figure 6.4: 6 equally spaced snapshots of a simulation rollout of the three-body spring-mass system. **(a)** shows velocity Verlet and **(b)** shows HyperVerlet. Both systems are computed using a step size of $h = 0.06$ whereas the ground truth has $h = 0.0003$. The tail following the bodies is a fading interpolation of the last 30 time steps.

shown in Figure 6.4b. By inspecting the frames at time 24 and 30, we can see small errors in the prediction starting to form, and if the duration was longer, the prediction would soon diverge from the ground truth.

6.2 Generalization study

An interesting question for machine learning models is how well they generalize when faced with variations in the input space they have been trained on. This section examines the generalizability of HyperVerlet when the time step varies across training, how it performs when the systems parameters are varied during training, and how it performs when introduced to samples outside the training distribution. The experimental setup follows the integrator comparison for the pendulum system with the largest step size $h = 0.08$ where the ground truth is obtained by a FR4 integrator with a step size $h = 0.001$.

To test the ability of HyperVerlet to generalize across different step sizes, we sample the step size h from a normal distribution $\mathcal{N}(\mu_h, \sigma_h)$ with a standard deviation (std.) $\sigma_h = \rho \cdot \mu_h$ as a percentage ρ of the mean step size μ_h. We conduct the experiment with the following percentages: $\rho \in [0\%, 10\%, 20\%, 30\%, 40\%]$. The sample mean μ_h is equal to 0.08, which is the largest step size in the integrator comparison for the pendulum described in Section 6.1.2.

On Figure 6.5a we see how the lowest MSE is achieved with a step size std. of 10%. Introducing a 10% variation to the step size during training allows the model to better learn the surrounding curvature, and thereby adding a regularizing effect during training, leading to better performance. From Figure 6.5b, we see an increase in VPT of 7.8% when introducing a variable step size of 10% into the training. However, if the step size is varied more than 10% we see a drop in VPT performance and an increase in MSE. A varied time step of 10% is 39.3% better on the VPT measure than a relative std. in time step of 20%.

To gain insight into the generalization with respect to task specific parameters, we conduct an experiment where we vary the task specific parameters within the same distribution under

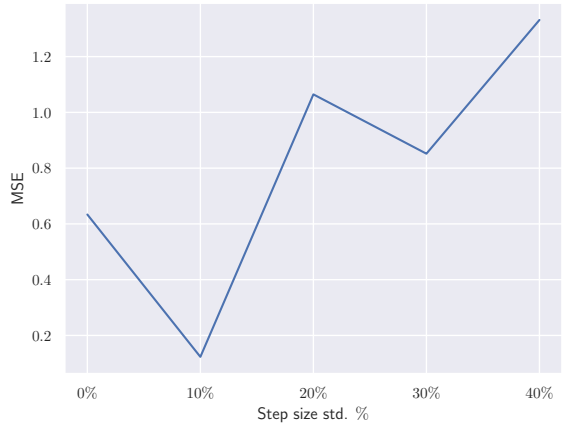
training and testing, i.e. interpolation. The experiment is conducted on the pendulum system where the length and mass of the pendulum is sampled from a Gaussian distribution $\mathcal{N}(1, 0.1)$ and $\mathcal{N}(0.9, 0.1)$ respectively. The result of this experiment are presented as box plots in Figure 6.5c and 6.5d showing the log(MSE) and VPT respectively for each configuration. From Figure 6.5c, we see how HyperVerlet when introduced to variation in the task specific parameters fails to conserve the total energy of the system and as a result, some configurations accumulates energy, thereby accelerating the pendulum bob, resulting in a high MSE. For this reason, we choose to depict the MSE as log(MSE). This accumulation of energy is also evident in Figure 6.5d where we can see that when the task specific parameters are not kept constant, the median VPT are halved compared to when they are constant.

To gain insight into the extrapolation capabilities of HyperVerlet, we conduct an experiment to see the performance when introduced to task specific parameters drawn from a different distribution than the training set. The results from this experiment can be seen in Figure 6.5e and 6.5f showing the log(MSE) and VPT respectively. From Figure 6.5e, we see that the MSE explodes for some configurations, which is a result of accumulation of energy. From Figure 6.5f, we see how the VPT for the test split decreases compared to that of training, hence HyperVerlet extrapolates poorly for longer durations. One thing to note is how the model is better at extrapolating over mass than over length. This can be seen from the box plot depicting the training where both the log(MSE) and VPT is better when extrapolating over masses rather then lengths. We contribute this to the fact that the mass has a linear relationship in the computation of position and velocity, whereas the length has a quadratic relation to the angular velocity as can be seen in Equation 5.8.

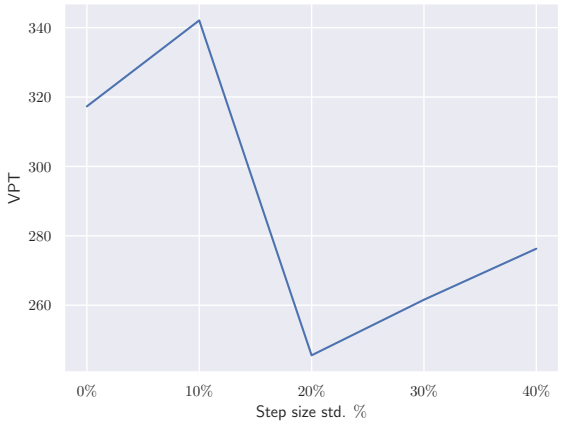
One thing to note about these results is how accumulation of energy penalizes the MSE much harder then dissipation of energy because there is no upper bound for how much kinetic energy the system can accumulate over time. To conclude, HyperVerlet does not generalize well across task specific parameters. To utilize the full potential of HyperVerlet, currently one has to train a specialized model with specific task variables.

6.3 Ablation study

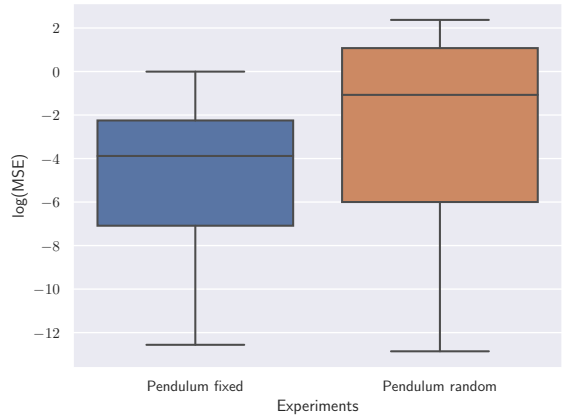
To gain insight into the importance of different features of HyperVerlet, we conduct an ablation study. An ablation study in this context is the alteration or removal of features like loss function, model structures, inputs, etc. The experimental setup follows the integrator comparison for the pendulum system with the largest step size $h = 0.08$ where the ground truth is obtained by a FR4 integrator with a step size $h = 0.001$. The parameters, i.e. pendulum length and gravity, are fixed to reduce the impact of randomness. We choose the pendulum system for the ablation study because it is non-linear and has no analytical solution but is sufficiently simple and more transparent than the three-body spring-mass system.



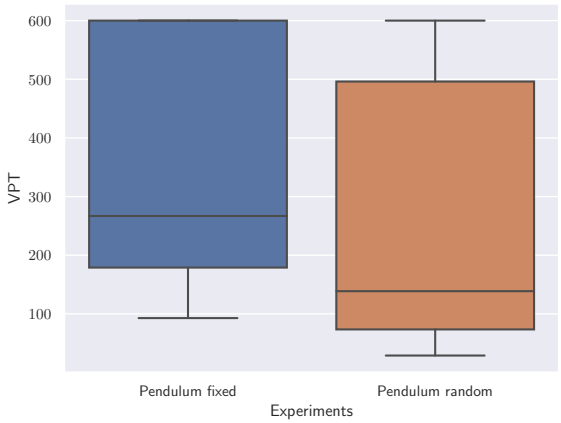
(a)



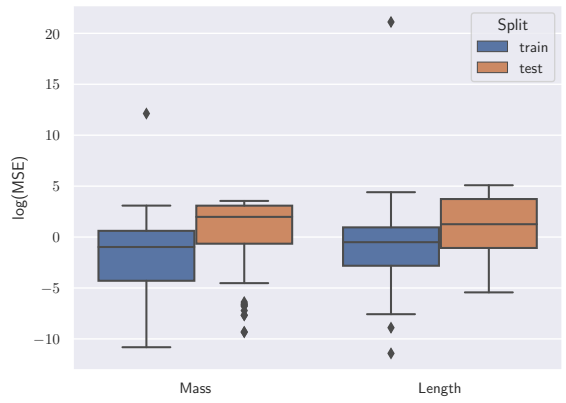
(b)



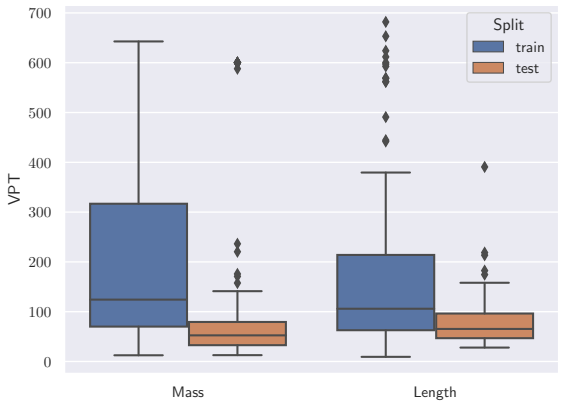
(c)



(d)



(e)



(f)

Figure 6.5: Generalization studies performed on the pendulum system. Figures **(a)**, **(c)**, and **(e)** show the MSE measure and figures **(b)**, **(d)**, and **(f)** show the VPT measure. The y-axis of **(c)** and **(e)** is log(MSE) because of poor performance on some outlier configurations. Figures **(a)** and **(b)** compares the performance when the step size is sampled from a Gaussian distribution $h \sim \mathcal{N}(\mu_h, \rho \cdot \mu_h)$ with ρ being the independent variable. Figures **(c)** and **(d)** compares fixed and randomized task specific parameters. Finally, figures **(e)** and **(f)** tests the extrapolation capabilities of HyperVerlet by comparing the performance on train and test datasets drawn from different distributions.

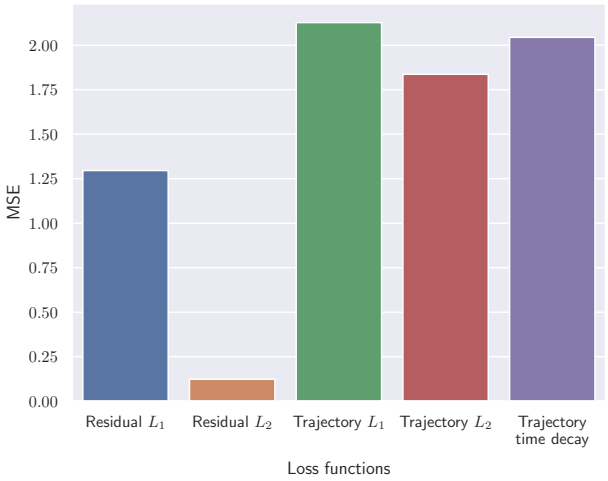


Figure 6.6: Comparison of accuracy measured by MSE when trained with different loss functions and fitting methods, i.e. residual and trajectory fitting. Smaller is better.

6.3.1 Loss function

First, we analyze the loss function as it has the largest impact on the results of HyperVerlet. Section 4.2 defined two families of loss functions for this domain: residual and trajectory fitting.

$$\mathcal{L}_{traj} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\hat{q}_n, q_n, \hat{v}_n, v_n) \tag{6.2}$$

$$\mathcal{L}_{res} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(g_\theta, \mathcal{R}) \tag{6.3}$$

with the number of time steps $N = 10$ for this study. The concrete loss functions belonging to either family depends on the choice of inner loss function $\mathcal{L}$. For this ablation study, we focus on Mean Absolute Error (MAE) or L_1 loss and MSE or L_2 loss, and for L_2 loss and trajectory fitting, we also analyze whether a exponential temporal discount factor γ improves the accuracy.

$$\mathcal{L}_{traj} = \frac{1}{N}\sum_{n=1}^{N}\gamma^n\mathcal{L}(\hat{q}_n, q_n, \hat{v}_n, v_n) \tag{6.4}$$

Since $N = 10$, we choose $\gamma = 0.9$, which yield a discount on the final time step of $\gamma^{10} = 0.349$. Time decay is not included for residual fitting as gradients are not backpropgated through time for this method and thus, the discount factor is meaningless.

The results of the loss function comparison can be seen in Figure 6.6. As is evident, residual fitting outperforms trajectory fitting by a wide margin. An explanation of this relationship is that residual training minimizes the local truncation error while trajectory fitting minimizes the global truncation error allowing a higher error at one time step to minimize the loss across all time steps. We conjecture that minimizing the local truncation error is better for generalization and this is why we see the lower loss for residual fitting. Finally, we conclude L_2 loss is better than L_1 loss and time decay does not improve trajectory fitting.

6.3.2 Corrector structure

While we have chosen a predictor-corrector architecture wrt. (q, p) for HyperVerlet to match the desired properties of the base solver, velocity Verlet, there exists other architectures that incorporate the corrector in other fashions. This section aims to explore the impact of this decision. Note that residual fitting is impossible for some of these methods due to their structure, so we use trajectory fitting throughout this study.

Figure 6.7a compares HyperVerlet to two other models; interleaving and alternating. The interleaving model is a modification of the 3-step form of the velocity Verlet algortihm (see Section 2.2.4) with a corrector for each step. We denote variables from velocity Verlet with tilde and variables from the corrector with hat.

$$\tilde{v}_{n+1/2} = v_n + \frac{h}{2} \cdot a(q_n) \tag{6.5}$$

$$\hat{v}_{n+1/2} = \tilde{v}_{n+1/2} + h^2 \cdot g_\theta^v(v_n, a(q_n), \tilde{v}_{n+1/2}, a(q_n), m, \Gamma) \tag{6.6}$$

$$\tilde{q}_{n+1} = q_n + h \cdot \hat{v}_{n+1/2} \tag{6.7}$$

$$\hat{q}_{n+1} = \tilde{q}_{n+1} + h^2 \cdot g_\theta^q(v_n, a(q_n), \hat{v}_{n+1/2}, a(\tilde{q}_{n+1}), m, \Gamma) \tag{6.8}$$

$$\tilde{v}_{n+1} = \hat{v}_{n+1/2} + \frac{h}{2} \cdot a(\hat{q}_{n+1}) \tag{6.9}$$

$$\hat{v}_{n+1} = \tilde{v}_{n+1} + h^2 \cdot g_\theta^v(v_n, a(q_n), \tilde{v}_{n+1}, a(\hat{q}_{n+1}), m, \Gamma) \tag{6.10}$$

The other architecture, alternating, adopts the 3-step form of velocity Verlet, $v \to q \to v$, for the corrector but maintains the predictor-corrector architecture. Assuming a velocity Verlet predictor $\tilde{q}_{n+1}, \tilde{v}_{n+1} = VV(q_n, v_n)$, the corrector architecture is as follows

$$\hat{v}_{n+1/2} = \tilde{v}_{n+1} + h^3 \cdot g_\theta^v(v_n, a(q_n), \tilde{v}_{n+1}, a(\tilde{q}_{n+1}), m, \Gamma) \tag{6.11}$$

$$\hat{q}_{n+1} = \tilde{q}_{n+1} + h^3 \cdot g_\theta^q(v_n, a(q_n), \hat{v}_{n+1/2}, a(\tilde{q}_{n+1}), m, \Gamma) \tag{6.12}$$

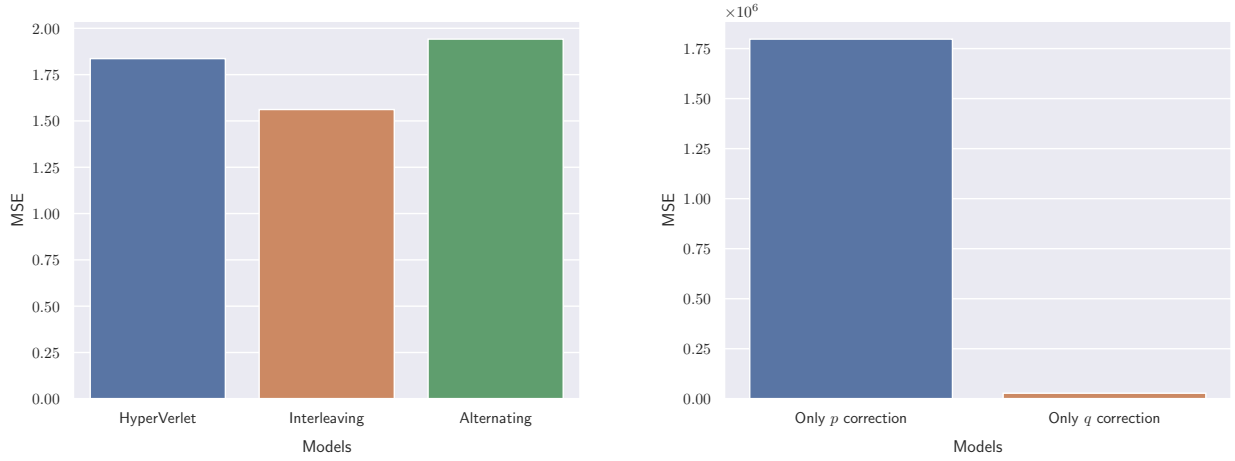
$$\hat{v}_{n+1} = \hat{v}_{n+1/2} + h^3 \cdot g_\theta^v(v_n, a(q_n), \hat{v}_{n+1/2}, a(\hat{q}_{n+1}), m, \Gamma) \tag{6.13}$$

Comparing HyperVerlet with these two architecture in Figure 6.7a, it seems as if the interleaving model is superior, but since residual fitting is impossible for both interleaving and alternating corrector architectures, HyperVerlet achieves the best performance.

The two simplest alterations are to only correct p or q, which due to the interacting nature of both variables for Hamiltonian systems still impacts the non-corrected variable. The results for these two methods can be seen in Figure 6.7b where one should note that the y-axis is $\times 10^6$. The obvious conclusion is that only correcting one of the variables is vastly inferior.

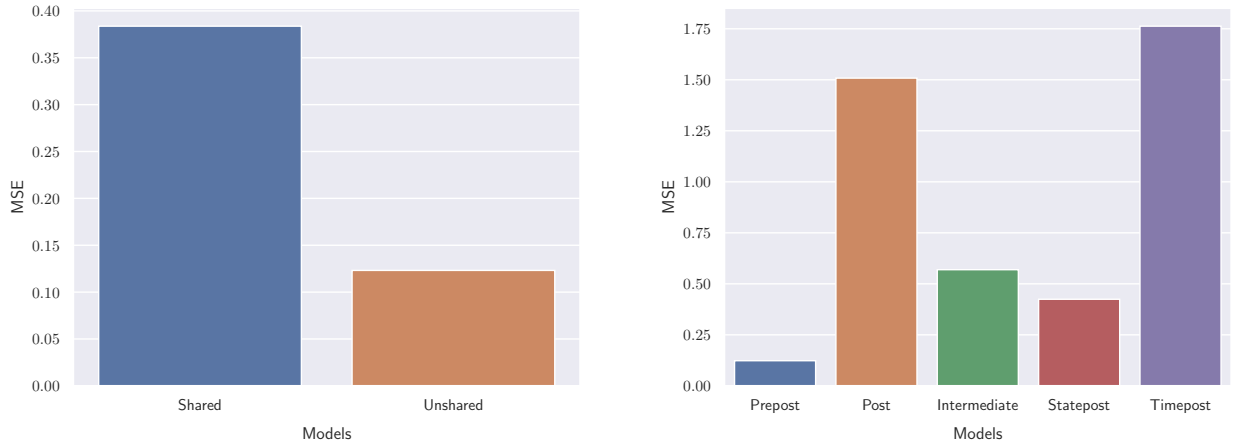
6.3.3 Shared vs unshared

In Section 4.1, we conjectured that shared layers between the q and p correctors, which can be interpreted as a feature extractor, will result in a deficient corrector. Literature has no consensus on this issue (Kipf et al., 2018; Chen et al., 2020; Sanchez-Gonzalez et al., 2020), but fortunately, this is a testable hypothesis and is the target of this ablation study. The shared and unshared models are equivalent fully-connected models except the shared model has a vector output for encapsulating both correctors. Figure 6.8a shows the results of this experiment where the unshared model is superior as expected.



(a) **(b)**

Figure 6.7: Comparison of different corrector architectures and their integration with the predictor. Note that all experiments are conducted using trajectory fitting because Interleaving and Alternating models cannot be trained with residual fitting. The plot is split in two because the models in **(b)** have a significantly higher MSE ($\times 10^6$). Smaller is better.



(a) **(b)**

Figure 6.8: Comparison of **(a)** the shared/unshared corrector architectures as described in Section 4.1 and **(b)** model input. The names for **(b)** are explained in Section 6.3.4. Prepost is HyperVerlet, but has been renamed to reflect the associated inputs. Smaller is better.

6.3.4 Model input

The final ablation study is an analysis of the impact different model inputs have, which can be interpreted as feature engineering. This analysis is important because the input has a huge impact on performance. The experiments are conducted on a pendulum system with an unshared corrector architecture, residual fitting, and MSE loss. Figure 6.8b shows the results of the experiment where the description below explains the inputs associated with each name. All models take mass and experiment specific parameters, i.e. pendulum length and gravity, as input.

Prepost The inputs for this model are the gradient vector fields of the phase space before and after the velocity Verlet update, $(v_n, a(q_n))$ and $(v_{n+1}, a(q_{n+1}))$. This model is HyperVerlet, but has been renamed for this experiment to reflect the associated inputs.

Post The inputs for this model are the gradient vector field of the phase space $(v_{n+1}, a(q_{n+1}))$ after the velocity Verlet. The expectation is that the reduced information about the step including phase space curvature and step size will negatively impact the performance.

Intermediate The inputs for this model are the gradient vector fields of the phase space before and after the velocity Verlet update, $(v_n, a(q_n))$ and $(v_{n+1}, a(q_{n+1}))$, and after each intermediate step of the velocity Verlet in 3-step form $(v_{n+1/2}, a(q_n))$, $(v_{n+1/2}, a(q_{n+1}))$. While this model has extended input compared to HyperVerlet and thus may improve the results, the redundant information may degrade the performance.

Statepost The inputs for this model are the state (q_{n+1}, v_{n+1}) and the gradient vector field of the phase space $(v_{n+1}, a(q_{n+1}))$ after the velocity Verlet update. Since the errors of the velocity Verlet does not depend on the canonical coordinates in the phase space but only the magnitude and direction of the gradient, we expect a degraded performance.

Timepost The inputs for this model are the gradient vector field of the phase space after the velocity Verlet update, $(v_{n+1}, a(q_{n+1}))$ along with the current time step and step size. This thesis only explores non-time dependent Hamiltonian systems, and thus, we expect this model to be inferior to HyperVerlet.

Looking at Figure 6.8b, it is evident that Prepost, or HyperVerlet, outperforms any other permutation of inputs. Additionally, we can conclude that information from before and after the velocity Verlet update is paramount to achieving good results because Post and Timepost have a significantly higher MSE. Observing Statepost, we also see inferior results, which we attribute to the fact the local truncation error induced by velocity Verlet does not depend on the canonical coordinate but the gradient. Finally, the hypothesis of redundant information for the Intermediate model is also observed to be true by Figure 6.8b.

Chapter 7

Discussion

HyperVerlet is a method to improve predictions of Hamiltonian systems for larger time steps compared to velocity Verlet. The theoretical analysis and empirical results supports this goal for long time horizons. Since HyperVerlet is not symplectic (see Section 4.3), energy may accumulate or dissipate, but for short to long durations, the empirical results show that the energy for HyperVerlet is more stable than for velocity Verlet, yielding better synchronization for system like spring-mass and pendulum. To ameliorate the issue of non-symplecitity, we can draw inspiration from SympNets (Jin et al., 2020) where the neural network is constrained to three layer types that are symplectic transformations. Since we choose velocity Verlet, which is symplectic (see Section 2.2.4), as the predictor, to make HyperVerlet symplectic, we only need to ensure that the corrector is symplectic, which is possible by the design of SympNets.

An important observation about HyperVerlet is its lack of generalization (see Section 6.2). An interpretation of this behavior is that the method is good at smoothing discrete time steps in a single phase space but the parameterization is insufficient for a familiy of phase spaces corresponding to varying masses and task specific parameters. This could either be a result of underparameterization, i.e. too few weights, or wrong inductive biases leading to subpar performance. If we are to constrain the corrector to a SympNet, the generalization problems are amplified as it is only parameterized to a single phase space. However, we have derived an extension to SympNets where the weights of the symplectic transformations are computed from the task specific parameters yielding a stronger model that generalizes to families of phase spaces. The mathematics of this extension can be found in Appendix C.1. For molecular dynamics, we also want scalability in the number of atoms, which is achievable with GNs as described in Section 4.1. Similarly, for a SympNet-based corrector such a model is desirable, and in Appendix C.2, we have derived a SympNet-like GN model.

The ultimate goal of HyperVerlet is to show a wall-clock speedup relative to traditional solvers, and especially velocity Verlet since it is so prevalent, by performing larger step sizes without sacrificing accuracy. However, the empirical results show that HyperVerlet is slower, but we attribute this to the fact that the systems are sufficiently simple and to compute the acceleration is inexpensive for these. Considering a complex system with an expensive force computation, it is desirable to minimize the number of these computations by doing larger time steps. Similarly to (Poli et al., 2020b), we can define the relative overhead O_r wrt. velocity Verlet in terms of multiply-accumulate operations (MACs).

$$O_r = \frac{MAC_{VV} + MAC_{net}}{MAC_{VV}} = 1 + \frac{MAC_{net}}{MAC_{VV}} \tag{7.1}$$

Considering a concrete scenario where velocity Verlet and the neural corrector require an equal amount of computation. In this case, HyperVerlet is only more efficient if it can achieve the same accuracy for at least twice the step size.

7.1 Modelling and comparison of chaotic systems

By definition, chaotic systems, like the three-body spring-mass system, are highly sensitive to changes in the initial condition, and also highly sensitive to errors during inference; especially

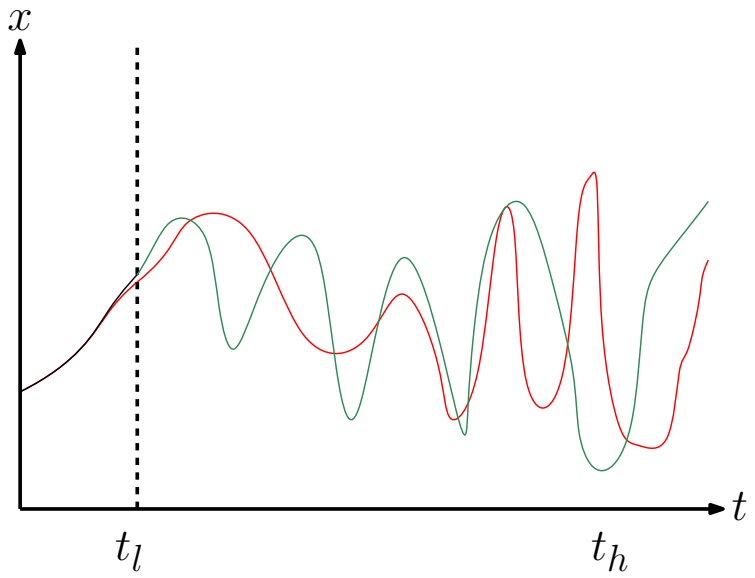


Figure 7.1: Conceptual figure illustrating the issue with MSE for chaotic systems. Let the green line be the ground truth and the red line be the prediction. The dashed line illustrates the point of rapid decline in correlation between the prediction and the ground truth. t_l and t_h on the t-axis illustrates a low and high MSE respectively.

because the errors accumulate by continuing the inference based on the predicted state. As a result, naive use of MSE as both a loss function and a measure may be undesirable. Starting by considering the case of using MSE as a loss and looking at Figure 7.1, one may attempt optimizing the neural network by minimizing the MSE between the prediction and the ground truth given the green and red trajectory respectively. An optimization across the entire trajectory would result in a high MSE loss due to the difference at time t_h, and thus, a large potential gain by adjusting the weights to better fit this point. However, adjusting the weights to reduce the loss at time t_h has little to no impact on the accuracy of the prediction. This is because the root cause of the error happens at t_l where the prediction and the ground diverge and thereby creating two separate trajectories.

From the perspective of measure, the issue is also present in our experiments as shown in Table 6.5. We see little difference between the performance of different solvers due to the fact that when the ground truth and the prediction split, the noise induced by the early accumulation of errors dominate the MSE. A concrete example of this issue can be seen by comparing the results of Heun and HyperHeun for the step size $h = 0.06$ in Table 6.5 and 6.6. Here, Heun achieves a lower MSE compared to HyperHeun but HyperHeun achieves a much longer VPT. While VPT shows the time until divergence of the prediction and ground truth, it has disadvantages of its own. VPT gives no insights into what went wrong or how realistic the trajectory is after its divergence. The measure also has the issue of the hyperparameter ϵ, which may be abused to hide the true performance of the method under study with no apparent process for parameter selection. To summarize, for simulations of chaotic systems, the traditional loss functions and measures may not be adequate, and no solution is immediately obvious.

Since we are dealing with Hamiltonian systems, we do know certain measurable properties that must be satisfied across the entire trajectory. As reiterated before, one such property is conservation of energy, and therefore, it may be tempting to incorporate energy into the loss function. But one cannot naively incorporate energy as the most trivial conservative transformation

of the phase space is to not progress time. Instead to force time progression and conservation of energy, we may compare the prediction and ground truth based on kinetic and potential energy individually.

7.2 Architectural interpretation and exploration

Property of interest to the problem but not explored in this thesis is time reversibility, or integrator symmetry. When applying the same integrator for both forward and backward simulations and the original initial coordinate is recoverable, then the integrator is symmetric. The property is desirable for some types of Hamiltonian simulations, and velocity Verlet and Forest-Ruth are both symmetric integrators (see Section 2.2.4 and 2.2.5). HyperVerlet, however, is not designed to be time reversible, but we can incorporate this property by two methods: data augmentation and structural inductive bias. With data augmentation, trajectories sampled from the ground truth are randomly reversed to encourage learning a step function robust to the time direction. This method imposes time reversibility as a soft constraint and thus does not provide any guarantees regarding the reversibility. The other method, structural inductive bias, encodes the bias into the structure of the neural network similar to the reversibility of TaylerNet (Tong et al., 2021) or SympNets (Jin et al., 2020). Since the bias is structurally encoded, the time reversibility is guaranteed but limits valid architectures.

The activation function is another non-trivial architectural choice in the context symplectic systems, as certain properties must be satisfied to maintain a universal approximation (Jin et al., 2020). While ReLU is prevalent in deep learning for its ability to avoid vanishing gradients, it is insufficient for this case. Our initial exploratory experiments showed that ReLU was inferior to sigmoid, which aligns with the theoretical analysis of (Jin et al., 2020).

An interesting direction of future architectural exploration is higher-order symplectic hypersolvers. Both (Shen et al., 2020) and (Poli et al., 2020b) propose higher-order hypersolvers but none explore these for unknown reasons. We expect their non-symplecticity to be the biggest inhibitor for achieving high-accuracy simulations of Hamiltonian systems, and a higher-order symplectic hypersolvers, e.g. a 4^{th}-order Hyper Forest-Ruth solver with a SympNets inspired corrector, may ameliorate this issue.

7.3 Practical issues

Since HyperVerlet in its current state lack generalization, though hopefully solvable with the extended SympNets, it is important to specialize the neural network to the system under study. This reduces the computational benefits of applying a HyperVerlet, but for certain classes of systems, we may avoid retraining a neural network completely. MD simulations with pairwise potential energy functions is one such class of high regularity (see Section 5.3) allowing neural network reuse. The idea can be applied as transfer or meta learning where a neural network is trained to minimize the error across a wide range of pair potential functions acting as a pretrained network that can be quickly fine-tuned to the specific conditions.

While conservation of energy appears prohibitively restrictive for practical applications, symplectic integrators also work for non-conserving systems with minor modifications. E.g. when conducting an physical experiment, it is almost impossible to conserve the energy, and as a

result, one usually controls the temperature instead, which is a related to the thermodynamic average of the kinetic energy (Vollmayr-Lee, 2020). Examples of modifications to velocity Verlet to allow a constant temperature include velocity scaling, i.e. controlling the kinetic energy, and using a modified Hamiltonian, e.g. Nosé-Hoover thermostats. Since the modifications are compatible with HyperVerlet, it may also improve this class of simulations.

Chapter 8

Conclusion

In this thesis, we have presented HyperVerlet, a hypersolver that is build on top of the commonly used velocity Verlet, a symplectic IVP solver for Hamiltonian systems. We show how HyperVerlet outperforms other state of the art hypersolvers like the HyperEuler and HyperHeun (Shen et al., 2020; Poli et al., 2020b) measured by a lower Mean Squared Error (MSE) and a higher Valid Prediction Time (VPT) on all dynamical systems subject to test in this thesis. We employ both MSE and VPT because accumulating errors when simulating Hamiltonian systems impair the benefit of either and makes predictions about the systems inherently difficult. We provide illustrations showing why MSE might not be the correct measure when working with chaotic systems, while describing the advantages and disadvantages of the VPT measure. What the ideal measure is remains unknown, and thus is an open research question. In addition to the empirical evidence that HyperVerlet is superior to other hypersolvers, we also contribute with a theoretical proof that shows that the truncation error for HyperVerlet is a constant factor lower than for velocity Verlet.

One thing to note is that HyperVerlet is not symplectic, like its base solver, velocity Verlet, but it can be modified to be symplectic, as described in Section 7, by restricting the transformation by the neural network to be symplectic in the style of SympNets (Jin et al., 2020). However, as can be seen in Section 6 that symplecticity may not always yield better results, as was evident from the desyncronization of velocity Verlet for some of the experiments, e.g. see Figure 6.1a and 6.3a. We know that symplectic integrators are better as the simulation duration approaches infinity. However, in this thesis we show how HyperVerlet is capable of simulating long time durations and still outperform its traditional symplectic solver, while maintaining a total energy within the amplitude of energy oscillation of velocity Verlet.

The HyperVerlet excels at specialized tasks, i.e fixed pendulum length and mass, showing the prospects of this kind of deep learning models. Although HyperVerlet beats state of the art hypersolvers, improvements can still be made. One of the current limitations of the HyperVerlet is the lack of generalization when introduced to variable task specific parameters, limiting its practical applications. Improvements to make HyperVerlet generalize better could pave the way for a powerful new family of tools for predicting the temporal evolution of dynamical systems, which is essential in a wide variety of applications like MD and cosmic simulations. We propose one such improvement expected to help with generalization: extended SympNets, which are useful for families of phase spaces. A description of extended SympNets can be found in Appendix C.1.

Other directions for future work may include incorporating more physical priors like time reversibility. We also conjecture that letting a neural network model Hamiltonian systems allows solving stiff systems more easily, and future work may test this hypothesis. Finally, exploration of higher-order symplectic hypersolvers present an interesting challenge that might improve performance for larger time steps.

Bibliography

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. 17, 18, 61

Michael Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo, 2017. 20

Michael Betancourt, Michael I. Jordan, and Ashia C. Wilson. On Symplectic Optimization, 2018. 20, 21

Venkatesh Botu and Rampi Ramprasad. Adaptive machine learning framework to accelerate ab initio molecular dynamics. *International Journal of Quantum Chemistry*, 2015. doi: 10.1002/qua.24836. 17

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018. 1, 18, 23

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on Learning Representations*, 2020. 19, 25, 46

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian Neural Networks, 2020. 1, 18, 23

George Dasoulas, Kevin Scaman, and Aladin Virmaux. Lipschitz Normalization for Self-Attention Layers with Application to Graph Neural Networks, 2021. 26

Stefan Doerr, Maciej Majewski, Adrià Pérez, Andreas Krämer, Cecilia Clementi, Frank Noe, Toni Giorgino, and Gianni De Fabritiis. TorchMD: A Deep Learning Framework for Molecular Simulations. *Journal of Chemical Theory and Computation*, 2021. ISSN 1549-9626. doi: 10.1021/acs.jctc.0c01343. 18

Albert Fässler. *Fast Track to Differential Equations*. Springer, 2019. 11, 12

Etienne Forest and Ronald D. Ruth. Fourth-order symplectic integration. *Physica D: Nonlinear Phenomena*, 1990. ISSN 0167-2789. doi: 10.1016/0167-2789(90)90019-L. 15

Amir Gholaminejad, Kurt Keutzer, and George Biros. ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2019. doi: 10.24963/ijcai.2019/103. 18

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 1, 18, 23

Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the Störmer-Verlet method. *Acta numerica*, 2003. 13, 14

Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks*, 2020. ISSN 0893-6080. doi: 10.1016/j.neunet.2020.08.017. 20, 28, 35, 49, 51, 53, 61

Michael I. Jordan. Dynamical, Symplectic, and Stochastic Perspective on Gradient-based Optimization. In *Proceedings of the International Congress of Mathematicians*, 2018. doi: 10.1142/9789813272880_0022. 20, 21

Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning Differential Equations that are Easy to Solve. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. 19

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural Relational Inference for Interacting Systems. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018. 25, 46

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning-accelerated computational fluid dynamics, 2021. 17

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable Gradients for Stochastic Differential Equations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research. PMLR, 2020. 18

Zijie Li and Amir Barati Farimani. Learning Lagrangian Fluid Dynamics with Graph Neural Networks, 2021. 17, 61

Marios Mattheakis, David Sondak, Akshunna S. Dogra, and Pavlos Protopapas. Hamiltonian Neural Networks for solving differential equations, 2020. 20

Isaac Newton. *Philosophiæ Naturalis Principia Mathematica*. William Dawson & Sons Ltd., London, 1687. 3, 4

Edward Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 2002. doi: 10.1017/CBO9780511803260. 1

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph Neural Ordinary Differential Equations, 2020a. 19

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward Fast Continuous-Depth Models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020b. 1, 2, 21, 22, 23, 25, 26, 49, 51, 53

Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. 18

Ronald D. Ruth. A canonical integration technique. *IEEE Trans. Nucl. Sci.*, 1983. doi: 10.1109/TNS.1983.4332919. 15

Steindor Saemundsson, Alexander Terenin, Katja Hofmann, and Marc Deisenroth. Variational Integrator Networks for Physically Structured Embeddings. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research. PMLR, 2020. 19

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian Graph Networks with ODE Integrators, 2019. 1, 18, 61

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2020. 17, 25, 46, 61

Samuel S. Schoenholz and Ekin D. Cubuk. JAX, M.D.: A Framework for Differentiable Physics, 2020. 18

Xing Shen, Xiaoliang Cheng, and Kewei Liang. Deep Euler method: solving ODEs by approximating the local truncation error of the Euler method, 2020. 1, 2, 21, 23, 25, 26, 51, 53

Endre Süli. *An introduction to numerical analysis*. Cambridge University Press, 2003. ISBN 1-107-13229-0. 9, 10, 26

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2013. 20

William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 1982. doi: 10.1063/1.442716. 13

J. R Taylor. *Classical mechanics*. University Science Books, 2005. 3, 7

Joseph L Taylor. *Foundations of analysis*, volume 18. American Mathematical Soc., 2012. 10

Yunjin Tong, Shiying Xiong, Xingzhe He, Guanghan Pan, and Bo Zhu. Symplectic neural networks in Taylor series form for Hamiltonian systems. *Journal of Computational Physics*, 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110325. 19, 51

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019. 1

Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Phys. Rev.*, 1967. doi: 10.1103/PhysRev.159.98. 1, 13, 59

Pantelis R Vlachas, Jaideep Pathak, Brian R Hunt, Themistoklis P Sapsis, Michelle Girvan, Edward Ott, and Petros Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 2020. 35

Katharina Vollmayr-Lee. Introduction to molecular dynamics simulations. *American Journal of Physics*, 2020. 31, 52

Andre Wibisono, Ashia C. Wilson, and Michael I. Jordan. A variational perspective on accelerated methods in optimization. *Proceedings of the National Academy of Sciences*, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1614734113. 20

Huan Xu and Shie Mannor. Robustness and generalization. *Machine Learning*, November 2011. doi: 10.1007/s10994-011-5268-1. 26

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *International Conference on Learning Representations*, 2020. 19

Appendix A

Lennard-Jones potential

For a more faithful molecular dynamics simulation than spring potential, a simple interatomic potential is the Lennard-Jones potential (Verlet, 1967), which can model inert gasses and is a component of many more complicated potentials. The energy and force functions can be seen in Figure 5.3 and are defined as follows

$$V(r_{ij}) = 4\varepsilon \left(\left[\frac{\sigma}{r_{ij}} \right]^{12} - \left[\frac{\sigma}{r_{ij}} \right]^{6} \right) \tag{A.1}$$

$$F(r_{ij}) = -\frac{24\varepsilon}{r_{ij}} \left(2 \left[\frac{\sigma}{r_{ij}} \right]^{12} - \left[\frac{\sigma}{r_{ij}} \right]^{6} \right) \tag{A.2}$$

where $r_{ij} = \|q_i - q_j\|$ is the distance between particles i and j. σ denotes the value of r_{ij} where potential energy is zero, and ε is the well-depth or minimum as can be seen in Figure 5.3. The first term is a steep repulsive force, i.e. close particles repulsive each other, and the second term is the attractive force across longer distances.

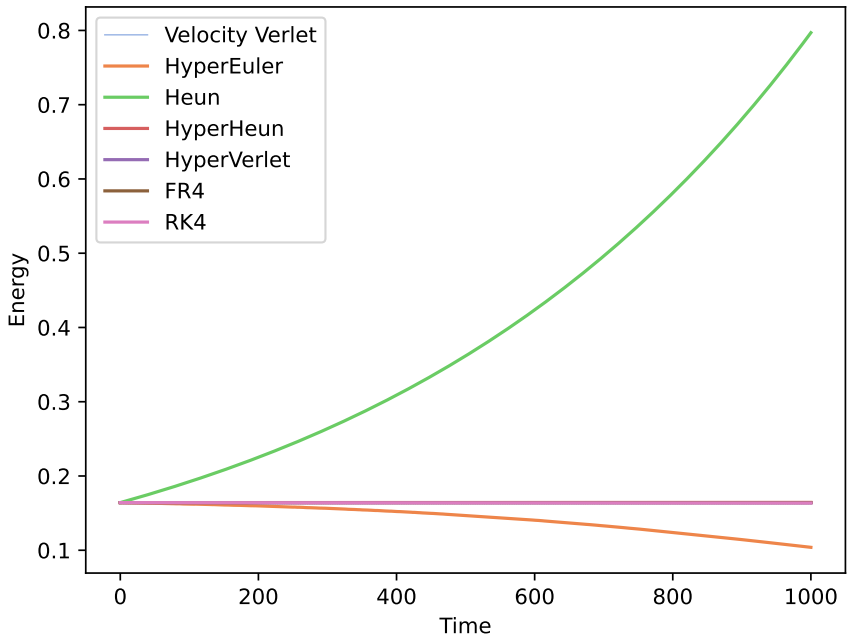
One example of a Lennard-Jones potential is the Kob-Andersen mixture, which is a mixture of two artifical particle types A and B in a ratio 80:20, because it can be used for modeling glass transitions. The Lennard-Jones parameters σ and ε depend on the types of particles interacting with units being defined relative to an interaction between two particles of type A, see Table A.1. All particles have a (unitless) mass of 1.

Table A.1: Lennard-Jones parameters for a Kob-Andersen mixture.

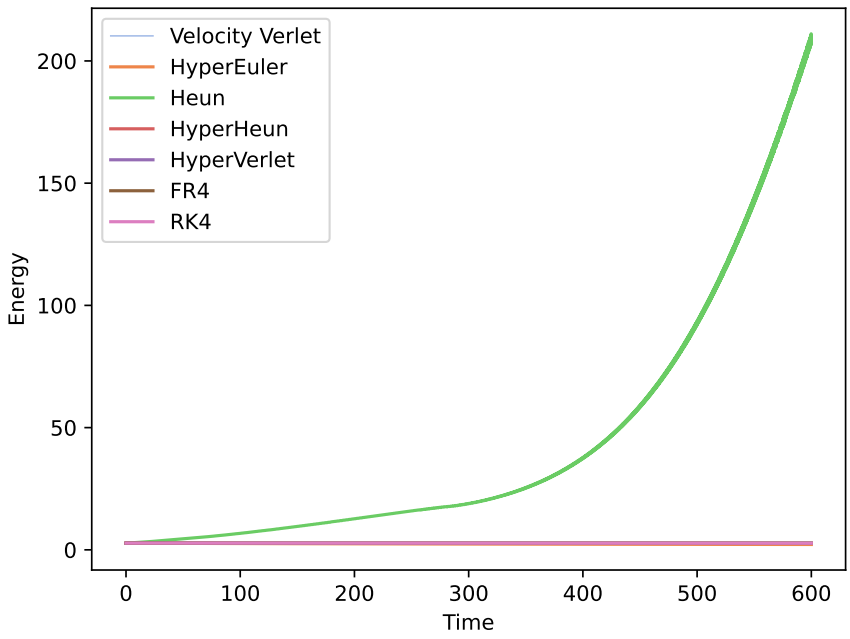
Particle types	σ	ε
AA	1.0	1.0
AB	0.8	1.5
BB	0.88	0.5

Appendix B

Total energy plots



(a)



(b)

Figure B.1: Both figures show the total energy of a single trajectory for a system for the full duration comparing the different solvers. Euler is excluded for clarity. (**a**) shows the total energy for the spring-mass system. (**b**) shows the total energy for the pendulum system.

Appendix C

SympNets

C.1 Parameter mapping

(Jin et al., 2020) describe their framework, SympNets, only consisting of symplectomorphisms, which can provably approximate all symplectic transformations. However, SympNets is not the parameterized by mass and task parameter specific, e.g. pendulum length and spring constant, which prohibits its generalization. We show an extension to two of their three layers, namely linear and activation layers or LA-SympNets, which can generalize to more task parameters.

Starting with the linear layer, which is a composition of linear symplectic modules and a bias

$$\mathcal{L}_n^{up}\left(\begin{bmatrix} q \\ p \end{bmatrix}\right) = \begin{bmatrix} I & 0/S_n \\ S_n/0 & I \end{bmatrix} \cdots \begin{bmatrix} I & 0 \\ S_2 & I \end{bmatrix} \begin{bmatrix} I & S_1 \\ 0 & I \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix} + b \tag{C.1}$$

$$\mathcal{L}_n^{low}\left(\begin{bmatrix} q \\ p \end{bmatrix}\right) = \begin{bmatrix} I & S_n/0 \\ 0/S_n & I \end{bmatrix} \cdots \begin{bmatrix} I & S_2 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ S_1 & I \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix} + b \tag{C.2}$$

where I is a d-dimensional identity matrix, $b \in \mathbb{R}^d$ and each $S_i \in \mathbb{R}^{d\times d}$ is symmetric, which is achieved by letting $S_i = A_i + A_i^T$ to allow unconstrained optimization. As is evident, one cannot input the mass or other task specific parameters, and as a result, the learned parameters A_i, b are specific to a single phase space. Our proposed extension is to let A be a function from the task parameters to a square matrix appropriate for computing S_i, parameterized by θ_i, and similarly for b to be a function from the task parameters to $\mathbb{R}^d$. That is, $A_i = A_{\theta_i}(m, \Gamma)$ and $b = b_\theta(m, \Gamma)$ where m is the mass and Γ are the task specific parameters. This architecture preserves the symplectic properties of this layer while generalizing L-SympNets to families of phase spaces. Additionally, there are no restrictions on the structure of A_{θ_i} and b_θ.

Similarly, the activation modules or A-SympNets can be generalized by letting the parameter vector $a = a_\theta(m, \Gamma)$ be a function of m, Γ parameterized by θ, while maintaining its symplecticity.

$$\Phi^{up}\left(\begin{bmatrix} q \\ p \end{bmatrix}\right) = \begin{bmatrix} p + diag(a) \cdot \sigma(q) \\ q \end{bmatrix} \tag{C.3}$$

$$\Phi^{low}\left(\begin{bmatrix} q \\ p \end{bmatrix}\right) = \begin{bmatrix} p \\ q + diag(a) \cdot \sigma(p) \end{bmatrix} \tag{C.4}$$

where σ is an element-wise activation function. Similarly to L-SympNets, there are no restrictions on the structure of a_θ, and we have derived a extension to LA-SympNets, which can generalize beyond a single phase space.

C.2 Graph networks

Since GNs are useful in many physical simulations (Battaglia et al., 2018; Sanchez-Gonzalez et al., 2020; Li & Farimani, 2021; Sanchez-Gonzalez et al., 2019), it is desirable to derive a symplectic flavor of this utilizing the structure of SympNets (Jin et al., 2020). We start by considering an undirected graph $G = (V, E)$ to ensure the symmetric property of S similar to the linear layer of SympNets. We let $n = |V|$ be the number of nodes and $q_v, p_v \in \mathbb{R}^f$ be the position and

momentum for a node v in an f-dimensional space. Then S is of size $nf \times nf$. Following the neural message passing scheme, let the incoming messages M_v to a node v wrt. a p update be

$$M_p^{up} = \{e_\theta(m_u, m_v, \Gamma(u,v)) \cdot q_u \mid u \in \mathcal{N}(v, G)\} \tag{C.5}$$

where m_u, m_v denotes the mass of u, v respectively and $\Gamma(u,v) = (\Gamma_u, \Gamma_v, \Gamma_{uv}, \Gamma_G)$ denotes task specific parameters for each node, the edge, and the graph. We assume that $e_\theta(m_u, m_v, \Gamma(u,v)) = e_\theta(m_v, m_u, \Gamma(v,u))^T$ to ensure the symmetry. $\mathcal{N}(v, G)$ denotes the neighborhood of node v in graph G, and we consider a node to be in the neighborhood of itself. We apply a sum to aggregate the incoming messages

$$p_v \leftarrow p_v + \sum M_v^{up} \tag{C.6}$$

The q update is computed by M_v^{low}, which is equivalent with M_v^{up} with q_u substituted with p_u.

This GN structure is equivalent with a symmetric S of an L-SympNet where any pair of nodes outside the neighborhood of each other has coefficients zero. Thus, the mapping is equivalent with a symplectic linear layer and is itself symplectic while maintaining the equivariant and scaling properties of GNs. Similarly to SympNets, we compose several of these message passing layers alternating between p and q update to improve the expressivity of the linear function.