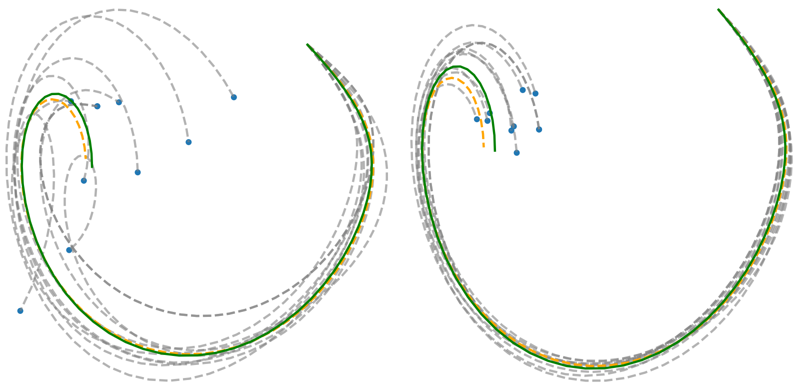
NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Håkon Noren

Numerical integration in inverse problems for ordinary differential equations

With application to Hamiltonian systems with noise in the observed data

Master's thesis in Industrial Mathematics
Supervisor: Elena Celledoni
Co-supervisor: Sølve Eidnes
June 2022



Håkon Noren

Numerical integration in inverse problems for ordinary differential equations

With application to Hamiltonian systems with noise in the observed data

Master's thesis in Industrial Mathematics
Supervisor: Elena Celledoni
Co-supervisor: Sølve Eidnes
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Preface

This thesis was written during the spring 2022 and concludes a five-year master's programme in Applied Physics and Mathematics at the Norwegian University of Science and Technology (NTNU) with a specialization in the field of Industrial Mathematics.

Working with this project, I have received supervision from Elena Celledoni (NTNU) and Sølve Eidnes (SINTEF Digital). I am grateful for the valuable input and critical questions you have asked, Elena. In addition, it was a great learning experience getting the chance to present the thesis work at the Manifolds and Geometric Integration Colloquia. Sølve, thanks for the patience and encouragement you have shown during our many digital meetings. Thanks also for the invitation to have a week-long stay at SINTEF in Oslo. This was a source of inspiration and motivation. Furthermore, it was encouraging to receive the SINTEF Master scholarship, supporting the work.

If it takes a village to raise a child, then my family is a village. Thanks for teaching me the value of hard work, kindness and curiosity. I am truly grateful for all the good friends I was fortunate to meet here in Trondheim and for numerous long coffee breaks with strange conversations at Matteland. My best friend and currently also my landlord, Ingrid, thanks for being such a good companion during the pandemic lockdown and for all the support.

If it takes a village to raise a child, it takes a well-functioning welfare state to educate a master student. Including a significant number of tax payers and a stable democracy. In the coming years, it is up to me and my fellow graduates to prove that we were worth the investment.

Trondheim, Norway
June 2022

Håkon Noren

Abstract

Inverse problems for ordinary differential equations (ODEs) aim at approximating the vector field given a set of points that are assumed to solve the ODE. Recently, research on using neural networks to solve inverse ODE problems on Hamiltonian form has gained traction. However, there has been little effort to systematically explore the space of different numerical integrators that could be used to solve this type of problems.

This thesis aims first at characterizing properties of numerical integrators that are beneficial when solving inverse ODE problems in general and problems on Hamiltonian form specifically. Mono-implicit Runge–Kutta (MIRK) methods are shown to be a class of integrators that are explicit for inverse problems. Here, we show how symmetric Runge–Kutta methods could be constructed by taking the mean of a MIRK method and its adjoint. Secondly, taking advantage of the inverse explicit property, a novel integration method called the mean inverse integrator, tailored for solving inverse problems with noisy data, is introduced. This method is proved to be less sensitive to noise in the data and this is verified in numerical experiments on multiple chaotic dynamical systems.

Samandrag

Dersom ein kjenner punkter av løysinga til ei ordinær differensiallikning (ODE), handlar det inverse problemet om å finne ein approksimasjon av vektorfeltet. Ei forskingsretning som nyleg har vorte særs aktiv, dreier seg om å nytte nevrale nettverk til å løyse inverse ODE problem på Hamiltonsk form. Trass dette, er det få som har gjort systematiske undersøkingar i kva numeriske integrasjonsmetodar som kan verte nytta i slike problem.

Denne tesa forsøker først å gi ein karakteristikk av nyttige eigenskapar ved numeriske integrasjonsmetodar i løysinga av inverse ODE problem. Først er generelle ODEar undersøkt, dernest ser vi på Hamiltonske system spesifikt. Mono-implisitte Runge–Kutta (MIRK) metodar viser seg å vere ei klasse integrasjonsmetodar som er eksplisitte for inverse problem. Her blir ei klasse symmetriske Runge–Kutta metodar utleia frå den adjungerte av MIRK metodar. Vidare, gjennom å nytte metodar som er invers eksplisitte, blir ein ny integrasjonsmetode, skreddarsydd for inverse problem med støy, introdusert. Det er mogleg å bevise at denne inverse integrasjonsmetoden er mindre sensitiv for støy i dataet og dette blir bekrefta i numeriske eksperiment utført på ulike kaotiske dynamiske system.

Contents

Chapter 1

Introduction

The ability to derive mathematical equations from empirical observations has been a driver of scientific progress for centuries. Equations describing the interaction of the objects taking part in a larger system could be used to predict future states and manipulate the system in order to control its dynamics. This thesis aims at studying methods for approximating functional terms in equations describing the dynamics of systems with energy preservation from a set of observations of the system. It is based on theory and methods from three different fields of study:

- Classical mechanics builds on the discoveries of Sir Isaac Newton aiming at deriving a more elegant formulation for the differential equations describing the dynamics of a mechanical system, focusing on how potential and kinetic energy changes with time.
- Secondly, geometric integration is the study of differential equations and their solutions, and how structures such as geometry or energy could be preserved over time. Predicting the motion of a pendulum, for instance, it is of importance to make sure that the length of the pendulum remains constant. Moreover, if there is no friction or damping of the pendulum, the energy of the predictions of the pendulum should remain constant as well.
- Finally, deep learning and artificial neural networks were studied by computer scientists already in the 1950s. However, it is only in the last decades that these methods have proved to have a significant ability to predict and approximate a range of different phenomena, such as classifying hand-written digits, generating realistic images of human faces or beating humans in chess or video games.

The dynamical systems studied in this thesis, are Hamiltonian systems which are ubiquitous in classical mechanics Here, the Hamiltonian is a scalar function, which when known, fully determines the dynamics of a mechanical system. In this thesis, we study how neural networks could be used to approximate the Hamiltonian from data. In particular, we investigate how numerical methods from geometric integration could be used to discretize the ODEs in time. The contributions of the thesis is the study and development of appropriate numerical integration methods that can be combined with neural networks to learn the Hamiltonian from data.

1.1 Structure of the thesis

Chapter 2 and 3 introduce the theoretical background for the thesis. The former states important results concerning numerical integration of Hamiltonian systems and the latter introduces neural networks and includes a review of research at the intersection of numerical integration and deep learning. The connection between mono-implicit Runge–Kutta methods and inverse ODE problems is discussed in Chapter 4. The chapter also includes a presentation and analysis of the mean inverse integrator, a method that approximates the discretized flow by computing the mean of multiple trajectories. Finally, numerical experiments in Chapter 5 puts the theory and the methods introduced to the test. The results are discussed in Chapter 6, before the thesis is concluded in Chapter 7.

1.2 Relation to the specialization project

The work presented in this master thesis builds on the theory studied, and code implemented working on the specialization project throughout the fall 2021. This project is described an unpublished report by Noren (2022). Here, symplectic integrators and discrete gradients were used to solve inverse ODE problems on Hamiltonian form. Inverse error analysis, and specifically modified equations were used to show that symplectic integrators guarantee the existence of a modified Hamiltonian that is the optimization target for the neural network in the training process. Furthermore, the report included a novel proof verifying the discrete gradient properties of a recent algorithm used to compute discrete gradients of neural networks. The discovery of the connection between mono-implicit Runge–Kutta methods and inverse problems, in addition to the mean inverse integrator, is novel work for this thesis.

Much of the theory introduced in Chapter 2, is a condensed version of what appeared in the report for the specialization project. Parts of Section 3.1 on neural networks are similar to this report, as well. Finally, the definition of the Hamiltonian systems found in Section 5.1.3 is the same as what is found in the specialization project.

Chapter 2

Geometric numerical integration

The following section presents how methods from geometric numerical integration could be used for application on Hamiltonian systems. First, some key concepts of numerical integration are introduced, before we define what a first integral is. Hamiltonian systems are then defined, together with a discussion on the symplectic property of their flow map, before introducing symplectic integrators and two classes of Runge–Kutta methods. Discrete gradients are another class of numerical integrators with beneficial preservation properties and are briefly introduced in the end. This chapter follows sections of the book Geometric Numerical Integration by Hairer et al. (2006), closely. Furthermore, the sections except the discussion of Runge–Kutta methods and implicit integration schemes summarize a lengthier discussion of the topics found in the report made during the specialization project, by Noren (2022).

2.1 Introduction to numerical integration

Numerical integration is the study of methods for calculating numerical values of integrals. This is particularly important when solving differential equations. The solution of an autonomous (f is not explicitly dependent on t) ODE could be found in time t_1 by

$$\begin{aligned} \frac{dy}{dt} &= f(y), \quad y(t_0) = y_0 \in \mathbb{R}^m \\ y(t_1) &= y_0 + \int_{t_0}^{t_1} f(y(t))\, dt. \end{aligned} \tag{2.1}$$

Here $y : [0, T] \to \mathbb{R}^m$, and for a given t we have $f : \mathbb{R}^m \to \mathbb{R}^m$. If we can solve or approximate the integral over the vector field f and know the initial value y_0, we can obtain an approximation of the solution to the ODE in the next time step t_1. ODEs can be solved by discretizing the time domain and approximating the integral.

Consider the time domain $\Omega_T = [0, T]$ and let the following notation define its discretization. Let

- N denote the number of subintervals that Ω_T is partitioned into.
- $h = \frac{T}{N}$ denote the length of each subinterval.
- $t_n = hn$ denote the grid points in time, for $n = 0, \ldots, N$
- $y_n \approx y(t_n)$ denote the numerical approximation on the grid.

Also note that in many cases, we write time derivatives as

$$\dot{y} := \frac{dy}{dt}.$$

The exact and the approximated solution of an ODE can be related to different flow maps.

Definition 2.1
Consider the exact solution of an ODE and a solution approximated by a numerical integrator. Let

$$\varphi_{h,f} : y(t_n) \to y(t_{n+1}), \tag{2.2}$$

denote the exact flow map and

$$\Phi_{h,f} : y_n \to y_{n+1}, \tag{2.3}$$

denote the discrete or numerical flow map of a given numerical integrator.

From this, we can define the *local error* and the *order* of an integrator, as defined by (Hairer et al., 2006, p. 29).

Definition 2.2
For any initial value $y_0 \in \mathbb{R}^m$ let

$$e(h) := \varphi_{h,f}(y_0) - \Phi_{h,f}(y_0), \tag{2.4}$$

be the local error.

Thus, we can define the order of a numerical integrator by how fast the local error decays when h tends to zero.

Definition 2.3
A one-step method has order p, if for all sufficiently regular ODEs, the local error satisfies

$$\varphi_{h,f}(y_0) - \Phi_{h,f}(y_0) = \mathcal{O}(h^{p+1}) \quad \text{as } h \to 0. \tag{2.5}$$

2.2 Conservation of first integrals

This subsection draws on (Hairer et al., 2006, Chapter IV) and introduces first integrals; quantities that are conserved along the trajectory of an ODE. Consider the differential equation

$$\dot{y} = f(y), \quad y(t) \in \mathbb{R}^m. \tag{2.6}$$

Definition 2.4 (First integrals)
A function $I : \mathbb{R}^m \to \mathbb{R}$ is called a first integral if

$$\frac{d}{dt} I(y) = 0, \quad \forall y(t) \text{ solving } (2.6).$$

By the chain rule, we see that this is equivalent to

$$\frac{d}{dt} I(y) = \nabla I(y)^T \frac{dy}{dt} = \nabla I(y)^T f(y) = 0.$$

Consider the class of quadratic first integrals that could be written on the form

$$Q(y) = y^T C y \quad C \in \mathbb{R}^{m \times m},\ C^T = C. \tag{2.7}$$

By Theorem 2.2 in (Hairer et al., 2006, Chapter IV.2), Runge—Kutta methods conserve $Q(y)$ if the coefficients satisfy

$$b_i a_{ij} + b_j a_{ji} = b_i b_j.$$

This will later be shown to be an important property when integrating Hamiltonian systems.

2.3 Hamiltonian systems

This section follows Hairer et al. (2006) and their introduction of Hamiltonian systems in Chapter VI.1. Hamiltonian systems are studied in classical mechanics, which is a continuation of the struggle of Newton to find mathematical formulations of the dynamics of mechanical systems.

Consider in general a physical system with the time dependent generalized coordinates $q(t) \in \mathbb{R}^d$. The Lagrangian of the system, due to Joseph-Louis Lagrange, is given by

$$L(q, \dot{q}) = T(q, \dot{q}) - U(q),$$

where T denotes the kinetic energy of the system, and U the potential energy. The Hamiltonian of the system is constructed by introducing the conjugate momentum p and the Legendre transformation on the form

$$\begin{aligned} p_k &:= \frac{\partial L}{\partial \dot{q}_k}, \qquad k = 1, \cdots, d, \\ H(q,p) &:= p^T \dot{q}(q,p) - L(q, \dot{q}(q,p)). \end{aligned} \tag{2.8}$$

We require that $\dot{q}$ can be expressed as a function of q, p and that this function is continuously differentiable and invertible. We now define Hamilton's equations by

$$\begin{aligned} \dot{q} &= \nabla_p H(q,p). \\ \dot{p} &= -\nabla_q H(q,p). \end{aligned} \tag{2.9}$$

Hamilton's equation (2.9), could be simplified by introducing $y = [q,p]^T$ and the canonical structure matrix

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

where $I_d \in \mathbb{R}^{d\times d}$ is the identity matrix. We then get Hamilton's equations on the form

$$\frac{dy}{dt} = J\nabla H(y). \tag{2.10}$$

It could be shown that $H(y)$ is a first integral by Definition 2.4, since

$$\frac{d}{dt}H(y) = \nabla H^T \dot{y} = \nabla H^T J \nabla H = 0.$$

Remark
Consider the flow map $\varphi_{h,f}(y_0)$ where $f(y) = J\nabla H(y)$ for Hamiltonian systems. To simplify notation, we will denote $\varphi_{h,H}(y_0)$ as the flow map of a Hamiltonian system, where the subscript H denotes the Hamiltonian.

A Hamiltonian system is separable if the Hamiltonian could be written as a sum

$$H(q,p) = H_1(q) + H_2(p). \tag{2.11}$$

This is indeed the case if we have kinetic and potential energy by $T(q,p) = T(p)$ and $U(q,p) = U(q)$.

2.4 Symplectic integration

The previous section introduced Hamilton's equations. In this section, we will introduce the concept of symplectic transformations. This property is a defining characteristic of Hamiltonian systems. Hence, by constructing symplectic numerical integrators, it is possible to preserves some of the structure of Hamiltonian systems.

2.4.1 Symplectic transformations

If y is the solution of a Hamiltonian system, a transformation ψ is symplectic, if and only if $z = \psi(y)$ is the solution of another Hamiltonian system. This is stated and proved in (Hairer et al., 2006, p. 187) and motivates the investigation of such transformations when Hamiltonian systems are investigated.

Definition 2.5
A linear mapping $A : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is called symplectic if

$$A^T J^{-1} A = J^{-1}$$
$$\textit{where } J^{-1} = -J \textit{ with } J = \begin{bmatrix} 0 & I_d \\ -I_d & 0 \end{bmatrix},$$

Which could be understood as a mapping preserving the sum of oriented areas, as presented in (Hairer et al., 2006, Ch. VI.2). A transformation in general is symplectic if its Jacobian matrix is a symplectic linear mapping.

Definition 2.6
A differentiable map $g : U \to \mathbb{R}^{2d}$, where $U \subset \mathbb{R}^{2d}$, is an open set, is everywhere symplectic if the Jacobian matrix $g'(q, p)$ is everywhere symplectic

$$g'(q, p)^T J^{-1} g'(q, p) = J^{-1}.$$

Theorem 2.4 (Hairer et al., 2006, p. 184) proves that the flow of a Hamiltonian system is symplectic and is given by

Theorem 2.7
Let the Hamiltonian $H(q, p)$ be twice continuously differentiable on $U \subset \mathbb{R}^{2d}$. For each fixed t, the flow $\varphi_{t,H}$ is a symplectic transformation.

Proof. The proof could be found in (Hairer et al., 2006, p. 184) and follows from the definition of the variational equation, the symmetry of the hessian $\nabla^2 H$, and by the fact that $J^{-1}J^T = -I$. □

2.4.2 Symplectic integrators

A symplectic integrator is then given by the following definition by (Hairer et al., 2006, p. 187).

Definition 2.8
A numerical one-step method is called symplectic, if the numerical flow

$$y_1 = \Phi_{h,f}(y_0), \tag{2.12}$$

is symplectic whenever the method is applied to a smooth Hamiltonian system.

Symplecticity could be studied by considering the variational equation of an ODE (Leimkuhler and Reich, 2005, Ch. 3.4) and conservation of quadratic invariants. Consider (Hairer et al., 2006, Chapter VI.4) which presents Lemma 4.1, a stepping stone to Theorem 4.3 in the same chapter, concluding that any integrator preserving quadratic invariants also preserve symplecticity. As mentioned in Chapter 2.2, a Runge–Kutta method does this if

$$b_i a_{ij} + b_j a_{ji} = b_i b_j.$$

Hence, such methods are symplectic integrators. Another approach to characterizing symplectic transformations and thus symplectic integrators could be found by using the wedge product, as presented in (Leimkuhler and Reich, 2005, Ch. 3.6).

2.5 Runge–Kutta methods

Following the notation of (Hairer et al., 2006, Ch. II.1.1), a Runge–Kutta method with s stages is a one-step numerical integrator given by

$$\begin{aligned} k_i &= f\Big(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_j\Big), \qquad i = 1,\dots,s, \\ y_{n+1} &= y_n + h\sum_{j=1}^{s} b_i k_i, \end{aligned} \tag{2.13}$$

and the method is specified by the coefficient matrix $A \in \mathbb{R}^{s\times s}$ and the vector $b \in \mathbb{R}^s$ where $a_{ij} = [A]_{ij}$, $b_i = [b]_i$, requiring that $c_i = \sum_{j=1}^{s} a_{ij}$ for $i = 1,\dots,s$. A method could be compactly represented by a *Butcher tableau* which structures the coefficients the following way:

$$\begin{array}{c|c} \text{c} & \text{A} \\ \hline & b^T \end{array}$$

A set of equations could be derived to determine the order of a given method. These equations are derived by comparing the Taylor expansion of the exact solution $y(t_{n+1})$ and the numerical approximation given by y_{n+1}. An elegant procedure to derive these equations exploit the rooted tree structure appearing when doing subsequent differentiation of the vector field $f(y)$ and this approach is presented in (Hairer et al., 2006, Ch. III.1.1).

2.5.1 Gauss collocation methods

The Gauss collocation methods could be shown to be symplectic and attain relatively high order for few stages. These methods are thus of interest when studying Hamiltonian systems. Collocation methods, following (Hairer et al., 2006, Ch. II.1.2), are derived by introducing a collocation polynomial that interpolates the solution of the ODE and its derivative in specified collocation points $\{c_i\}_{i=1}^{s}$. This enables a continuous approximation to the solution, and it is possible to show that these methods does belong to the class of Runge–Kutta methods. As stated in (Hairer et al., 2006, Ch. II.1.2), we have that:

Definition 2.9 (Collocation methods)
Let $\{c_i\}_{i=1}^{s}$ be distinct real numbers. The collocation polynomial $u(t)$ of degree s interpolates the ODE solution by

$$\begin{aligned} u(t_n) &= y_n \\ \dot{u}(t_n + c_i h) &= f\Big(t_n + c_i h, u(t_n + c_i h)\Big) \end{aligned} \tag{2.14}$$

such that the next integration step is found by

$$y_{n+1} := u(t_n + h)$$

As presented in (Hairer et al., 2006, Ch. II.1.2), using Lagrange interpolation in the points $\{c_i\}_{i=1}^s$, the collocation polynomial could be found by

$$\begin{aligned} \dot{u}(t_n + \tau h) &= \sum_{j=1}^{s} \dot{u}(t_n + c_i) l_j(\tau), \\ \text{where} \quad l_j(\tau) : &= \prod_{i \neq j} \frac{\tau - c_i}{c_j - c_i}. \end{aligned}$$

By integration over $\dot{u}(t_n + \tau h)$ and denoting $k_i := \dot{u}(t_n + c_i)$ we find

$$u(t_n + c_i h) = y_n + h \sum_{j=1}^{s} k_j \int_0^{c_i} l_j(\tau)\, d\tau.$$

Comparing this form to the Runge–Kutta methods by Equation (2.13), it is evident that the coefficients could be identified as

$$a_{ij} := \int_0^{c_i} l_j(\tau)\, d\tau \quad \text{and} \quad b_i := \int_0^1 l_i(\tau)\, d\tau.$$

By specifying the Lagrange interpolation polynomial and showing how this yields coefficients A and b, the remaining free variables for specifying the collocation method are the interpolation coefficients c_i. The Gauss collocation methods (Hairer et al., 2006, Ch. II.1.3) are defined by selecting c_i as the s roots of the shifted Legendre polynomial

$$\frac{d^s}{dx^s}\Big(x^s(x-1)^s\Big).$$

This yields an interpolation polynomial that interpolates polynomials of order $2s$ exactly, where $2s$ could be denoted as the interpolation order. Theorem 1.5 in (Hairer et al., 2006, Ch. II.1.2) proves that the integration order, by Definition 2.3, of the collocation method coincides with the interpolation order. Hence, Gauss collocation methods has order $p = 2s$.

Consider again the quadratic first integrals defined by Equation (2.7) and the proof of Theorem 2.1 in (Hairer et al., 2006, Ch. IV.2.1). The quadratic first integral is given by $Q(y) = y^T C y$ such that an insertion of the solution by a Gauss interpolation polynomial $y_{n+1} = u(t_n + h)$ gives us

$$\begin{aligned} Q(u(t_n + h)) &= Q(u(t_n)) + \int_{t_n}^{t_{n+1}} \frac{d}{dt} Q(u(t))\, dt \\ \iff \qquad y_{n+1}^T C y_{n+1} &= y_n^T C y_n + 2 \int_{t_n}^{t_{n+1}} u(t)^T C \dot{u}(t)\, dt \end{aligned}$$

Since $\frac{d}{dt} Q(u(t)) = 2u(t)^T C \dot{u}(t)$ by the symmetry of C and by definition we have that $u(t_n) = y_n$. Since $u(t)$ by definition is a polynomial of degree s, $u(t)^T C \dot{u}(t)$ is a polynomial of degree $2s-1$ and is integrated exactly by an s-stage Gaussian quadrature. Since

$$u(t_n + c_i h)^T C \dot{u}(t_n + c_i h) = u(t_n + c_i h)^T C f(u(t_n + c_i h)) = 0$$

by the definition of a quadratic first integral, the integrand is zero and the first integral is preserved since

$$y_{n+1}^T C y_{n+1} - y_n^T C y_n = 0.$$

Hence, Gauss collocation methods preserve quadratic invariants and are thus symplectic. For $s = 1$, we get the implicit midpoint method of order $p = 2$ and the two next methods with two and three stages are presented below.

Example 2.10 (Gauss collocation for $s = 2, 3$)
For $s = 2$ we find the following Gauss collocation method of order $p = 4$,

$$\begin{array}{c|cc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

whereas for $s = 3$ the method of order $p = 6$ is given by:

$$\begin{array}{c|ccc}
\frac{1}{2} - \frac{\sqrt{15}}{10} & \frac{5}{36} & \frac{2}{9} - \frac{\sqrt{15}}{15} & \frac{5}{36} - \frac{\sqrt{15}}{30} \\
\frac{1}{2} & \frac{5}{36} + \frac{\sqrt{15}}{24} & \frac{2}{9} & \frac{5}{36} - \frac{\sqrt{15}}{24} \\
\frac{1}{2} + \frac{\sqrt{15}}{10} & \frac{2}{9} + \frac{\sqrt{15}}{15} & \frac{5}{36} + \frac{\sqrt{15}}{30} & \frac{5}{36} \\
\hline
 & \frac{5}{18} & \frac{4}{9} & \frac{5}{18}
\end{array}$$

2.5.2 Mono-implicit Runge–Kutta methods

Certain classes of implicit Runge–Kutta methods aim at maintaining high order and good stability properties, while at the same time ensuring that the non-linear equations for computing the next step, are computationally less demanding to solve. The diagonally implicit Runge–Kutta methods (DIRK) (Wanner and Hairer, 1996, Ch. IV.6) is an example of one such class. Mono-implicit Runge–Kutta methods are another, which properties are discussed by Burrage et al. (1994). These methods turn out to be explicit in inverse ODE problems and are thus of particular interest for the problems studied in this thesis. They could be defined in the following manner:

Definition 2.11 (Mono-implicit Runge–Kutta methods)
Let $b \in \mathbb{R}^s$ be a coefficient vector for an s-stage Runge–Kutta method and $D \in \mathbb{R}^{s\times s}$ be a strictly lower triangular matrix. The s-stage Runge–Kutta method defined by D, b and the coefficient vector $v \in \mathbb{R}^s$ yielding the coefficient matrix

$$A := D + vb^T,$$

is a mono-implicit Runge–Kutta (MIRK) method.

Here $vb^T \in \mathbb{R}^{s\times s}$ is the outer product of the two vectors. As seen in Burrage et al. (1994), these methods could be re-written in a useful form

Lemma 2.12
The stages k_i of a MIRK method could be written as

$$k_i = f\Big(y_n + v_i(y_{n+1} - y_n) + h\sum_{j=1}^{s} d_{ij}k_j\Big), \tag{2.15}$$

where $d_{ij} := [D]_{ij}$.

Proof. Let $K = [k_1, \dots, k_s] \in \mathbb{R}^{m\times s}$ be a matrix with the intermediate stages and $a_i := [A]_i^T = [a_{i1}, \dots, a_{is}]^T \in \mathbb{R}^s$ be a transposed row of the coefficient matrix A. Then, a Runge–Kutta method could be written on the form

$$\begin{aligned} k_i &= f(y_n + hKa_i), \\ y_{n+1} &= y_n + hKb. \end{aligned}$$

For a MIRK method, we have that $a_i = d_i + v_i b$ and since $y_{n+1} - y_n = hKb$ we find that

$$\begin{aligned} k_i &= f\Big(y_n + v_i hKb + hKd_i\Big) \\ &= f\Big(y_n + v_i(y_{n+1} - y_n) + h\sum_{j=1}^{s} d_{ij}k_j\Big). \end{aligned}$$

□

The MIRK methods are usually represented by an extended Butcher tableau with an extra column for the v coefficient vector and the strictly lower triangular matrix D replaces the A matrix, yielding

$$\begin{array}{c|c|c} \mathrm{c} & \mathrm{v} & \mathrm{D} \\ \hline & & b^T \end{array}$$

In Burrage et al. (1994) it is proved that the maximum order of an s-stage MIRK method is $p = s + 1$ and several methods with stages $s \leq 5$ are presented. Consider two of these methods:

Example 2.13 (MIRK methods order $p = 3$ and $p = 4$)
Examples of MIRK methods with stages $s = 2, 3$ and order $p = 3, 4$ are given below, and are found by Burrage et al. (1994). The method with two stages is found by choosing $c = 1$ and the three stage method by $c = 0$, when considering the Butcher tableaux by Burrage et al. (1994). Both methods are A-stable.

$$\begin{array}{c|c|cc} 1 & 1 & 0 & 0 \\ \frac{1}{3} & \frac{5}{9} & -\frac{2}{9} & 0 \\ \hline & & \frac{1}{4} & \frac{3}{4} \end{array} \qquad \begin{array}{c|c|ccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{8} & -\frac{1}{8} & 0 \\ \hline & & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array}$$

2.6 Implicit integration schemes

The Gauss collocation methods have stages that are implicitly defined, meaning that a system of equations has to be solved for computing the stage value k_i. Similarly, the next step of a MIRK method requires solving a system for y_{n+1}. This section discusses how to solve such systems in both of these two cases, efficiently.

2.6.1 Equations for general implicit Runge–Kutta systems

This section follows (Sanz-Serna and Calvo, 2018, Ch. 5.5) and (Hairer et al., 2006, Ch. VII.6). Let now a Runge–Kutta method be represented by the stage values Y_i such that $k_i = f(Y_i)$, in comparison to how a Runge–Kutta method was given in Equation (2.13). The method is thus given by

$$\begin{aligned} Y_i &= y_n + h\sum_{j=1}^{s} a_{ij} f(Y_j) \quad i = 1, \dots, s, \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i f(Y_i). \end{aligned} \tag{2.16}$$

Without any assumptions on the coefficients A, computing Y_i requires solving a system of equations. Following (Sanz-Serna and Calvo, 2018, Ch. 5.4.1), we consider the increments

$$Z_i := Y_i - y_n,$$

enabling the rewriting of the stage computations in Equation (2.16) to

$$Z_i = h\sum_{j=1}^{s} a_{ij}(y_n + Z_j) \quad i = 1, \dots, s. \tag{2.17}$$

Considering the whole system, we could write

$$\underbrace{\begin{bmatrix} Z_1 \\ \vdots \\ Z_s \end{bmatrix}}_{:=Z} = h(A \otimes I_m) \underbrace{\begin{bmatrix} f(y_n + Z_1) \\ \vdots \\ f(y_n + Z_s) \end{bmatrix}}_{:=F(Z)},$$

where we introduce the vectors $Z, F(Z) \in \mathbb{R}^{sm}$ such that we can write the system of equations as

$$Z = h(A \otimes I_m)F(Z). \tag{2.18}$$

Note that $\otimes$ denotes the tensor product, and $I_m \in \mathbb{R}^{m\times m}$ is the identity matrix, such that

$$A \otimes I_m := \begin{bmatrix} a_{11}I_m & \dots & A_{1s}I_m \\ \vdots & & \vdots \\ a_{s1}I_m & \dots & a_{ss}I_m \end{bmatrix} \in \mathbb{R}^{sm\times sm}. \tag{2.19}$$

This allows solving one system with dimension sm instead of solving s systems with dimension m. In general, evaluations of the vector field f is computationally expensive, and in particular when f is given by a neural network. When the system in Equation (2.18) is solved for Z, we still would have to do s evaluations to compute y_{n+1}. However, following Sanz-Serna and Calvo (2018), it could be shown that this could be avoided, if we assume that the coefficient matrix A is invertible.

Consider first the following properties for the tensor product Van Loan (2000), that

$$\begin{aligned}(A \otimes B)(C \otimes D) &= (AC \otimes BD),\\ (A \otimes B)^{-1} &= (A^{-1} \otimes B^{-1}),\\ \text{and} \quad (A \otimes B)^T &= (A^T \otimes B^T).\end{aligned}$$

The implicit system by Equation (2.18) could be rewritten as

$$\begin{aligned}Z &= h(A \otimes I_m)F(Z)\\ (b \otimes I_m)^T(A^{-1} \otimes I_m)Z &= h(b \otimes I_m)^T F(Z)\\ (b^T A^{-1} \otimes I_m)Z &= h(b \otimes I_m)^T F(Z),\end{aligned}$$

which means we can avoid one evaluation of the vector field as we can find

$$\begin{aligned}y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i f(Y_i)\\ &= y_n + h(b \otimes I_m)^T F(Z)\\ &= y_n + (b^T A^{-1} \otimes I_m)Z,\end{aligned}$$

using the solution Z directly. Introducing $d := b^T A^{-1}$ we solve the implicit step by

$$\text{solve} \quad Z = h(A \otimes I_m)F(Z), \tag{2.20}$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} d_i Z_i. \tag{2.21}$$

As discussed in (Hairer et al., 2006, Ch. VIII.6.1) interpolation polynomials, such as the collocation polynomials in the case of Gauss collocation methods, could be used to obtain a more accurate initial guess starting the iterations for solving the system above. However, in this setting, we will use the simple initial guess by

$$Z_i^0 = hc_i f(y_n)$$

where $c_i = \sum_{j=1}^{s} a_{ij}$.

Remark

Consider the Gauss collocation methods with stages $s = 2, 3$ presented in Example 2.10. In both cases the coefficient matrix A is invertible, and we find for the method with $s = 2$, that

$$d = [-\sqrt{3},\ \sqrt{3}]^T,$$

and for $s = 3$ we find

$$d = [\frac{5}{3},\ -\frac{4}{3},\ \frac{5}{3}]^T.$$

2.6.2 Equations for mono-implicit Runge–Kutta systems

For MIRK methods, on the other hand, the stages only depend on y_n, y_{n+1}, and the integration scheme could thus be formulated as an implicit system for y_{n+1} only. In that case, the method is given by

$$\begin{aligned} Y_i &= y_n + v_i(y_{n+1} - y_n) + h\sum_{i=1}^{s} d_{ij} f(Y_j), \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i f(Y_i). \end{aligned}$$

Let us introduce the notation $\hat{F}(y_n, y_{n+1}) := \sum_{i=1}^{s} b_i f(Y_i)$ and the increment

$$z_n := y_{n+1} - y_n,$$

where the increment, and thus the next step is found by solving

$$\begin{aligned} z_n &= h\hat{F}(y_n, z_n + y_n), \\ y_{n+1} &= z_n + y_n. \end{aligned} \tag{2.22}$$

For starting iterations solving this scheme, we will simply take a step with explicit Euler such that

$$z_n^0 = hf(y_n).$$

2.7 Discrete gradients

The discrete gradient methods are another class of numerical integrators distinct from Runge–Kutta methods and are designed to preserve invariants of ODEs, where the Hamiltonian of Hamiltonian systems is one example. There are several specific examples of different discrete gradient methods, consider for instance the average vector field discrete gradient presented by McLachlan et al. (1999), or the midpoint discrete gradient discussed by Gonzalez (1996). Below, we will in a short manner state the main properties of the class of discrete gradient methods.

Consider the canonical Hamiltonian equation

$$\dot{y} = J\nabla H(y) \quad y(0) = y_0 \in \mathbb{R}^{2d} \tag{2.23}$$

$$J \in \mathbb{R}^{2d\times 2d}, \quad J = \begin{bmatrix} 0 & I_d \\ -I_d & 0 \end{bmatrix}. \tag{2.24}$$

In section 2.3 we showed that for such systems the Hamiltonian $H : \mathbb{R}^{2d} \to \mathbb{R}$ is a first integral, meaning that $H(y(t)) = H(y(t_0))$ is conserved along solution curves. Discrete gradients allow for preserving such invariants exactly. Given a first integral H, we define a discrete gradient by:

Definition 2.14
A discrete gradient $\overline{\nabla} H : \mathbb{R}^{2d} \times \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ *is a function that satisfies*

$$\begin{aligned} \overline{\nabla} H(x,y)^T (y-x) &= H(y) - H(x), \\ \overline{\nabla} H(x,x) &= \nabla H(x), \end{aligned} \tag{2.25}$$

for all $x, y \in \mathbb{R}^{2d}$.

This could be utilized to obtain numerical integrators of a certain order, as stated in the next theorem, which is presented and proved by Eidnes (2022).

Theorem 2.15 (Order of discrete gradients)
Let $\overline{\nabla} H(x,y)$ *be a discrete gradient by Definition* (2.14) *and continuously differentiable in its second argument. The discrete gradient method defined by*

$$y_{n+1} := y_n + hJ\overline{\nabla} H(y_n, y_{n+1}), \tag{2.26}$$

is consistent. If the Jacobian $\frac{\partial}{\partial y}\overline{\nabla} H(x,y)$ *of the discrete gradient is symmetric, the integrator is of order* $p = 2$.

It is trivial to show that such integrators preserve the invariant by considering

$$\begin{aligned} H(y_{n+1}) - H(y_n) &= \overline{\nabla} H(y_n, y_{n+1})^T (y_{n+1} - y_n) \\ &= h\overline{\nabla} H(y_n, y_{n+1})^T J \overline{\nabla} H(y_n, y_{n+1}) \\ &= 0. \end{aligned}$$

Furthermore, in Eidnes (2022), a methodology is derived for identifying the order conditions for methods of arbitrarily high order. In the setting of this thesis, the algorithm introduced by Matsubara et al. (2019), for computing discrete gradients of neural networks, is of particular interest. In the specialization project, the discrete Jacobian, as a generalization of the discrete gradient, was introduced and a novel proof was provided of how the algorithm by Matsubara et al. (2019) allowed for computing the discrete Jacobian of a neural network. Here, this algorithm will be used without further introduction.

Chapter 3

Deep learning and numerical integration

This section first defines what a neural network is, followed by a brief summary of recent research at the intersection of deep learning, numerical integration and ODEs.

3.1 Neural networks

A neural network is a parametrized function built by compositions of linear and non-linear transformations. It could be understood as a generalization of linear regression to the non-linear domain. As in regression, the aim is to take a finite set of observations, learn a mapping which allows for predicting some class membership or an unknown quantity. Where linear regression computes the model parameters by solving a least-squares optimization problem, the high dimensionality of the parameter space of neural networks, results in a complicated optimization problem. In general, the following have to be defined to obtain a functioning neural network:

1. Defining the neural network architecture by choosing how many parameters θ to include and how to structure a function (forward pass) from the parameters.

2. Defining how to compute gradients of the neural network output (or loss) with respect to all parameters θ (backwards pass).

3. Specifying how to utilize the training data or observations to search for the optimal parameters θ (optimization procedure).

One way to solve the first problem is by defining the forward pass of what is often called a multilayer perceptron (Goodfellow et al., 2016, Part II).

Definition 3.1
Let the neural network $F : \mathbb{R}^{n_0} \to \mathbb{R}^{n_N}$ be defined by the following recursion

$$\begin{aligned}
\tilde{a}_i &:= \tilde{f}_i(a_{i-1}) := W_i a_{i-1} + b_i \\
a_i &:= f_i(\tilde{a}_i) := \sigma_i(\tilde{a}_i)
\end{aligned}$$

$$\begin{aligned}
\tilde{f}_i &: \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, \quad f_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i} \\
W_i &\in \mathbb{R}^{n_i \times n_{i-1}}, \quad b_i \in \mathbb{R}^{n_i}
\end{aligned}$$

$$\begin{aligned}
\textit{where} \quad a_0 &= x \in \mathbb{R}^{n_0} \\
F(x) &:= a_N \in \mathbb{R}^{n_N},
\end{aligned}$$

assuming that n_i are strictly positive integers, $i = 1, \ldots, N$, and σ_i is a non-linear differentiable function applied element wise on $\tilde{a}_i$. Equivalently the neural network could be written as

$$F(x) = f_N \circ \tilde{f}_N \circ f_{N-1} \circ \tilde{f}_{N-1} \circ \cdots \circ f_1 \circ \tilde{f}_1(x).$$

Let matrices W_i and vectors b_i, $i = 1, \ldots, N$ be denoted as the set of parameters θ. The neural network could now be represented by $F_\theta(x) = F(x)$.

The second problem, is solved by the backpropagation algorithm (Goodfellow et al., 2016, Chapter 6.5), a special case of automatic differentiation. Automatic differentiation is a framework for breaking up complicated computational graphs into simple elementary operations and using the chain rule to compute gradients. For implementing the backpropagation of gradients, one has to define how to handle derivatives in each layer and how to store function evaluations of the forward pass in memory. Using the chain rule on the function compositions described in Definition 3.1 thus enables the computation of the gradients of the neural network output, or its loss, with respect to the various parameter vectors and matrices.

The optimization procedure requires specifying an optimization objective, or what is commonly called a loss function. Assume that we want to use a neural network f_θ to approximate a function $f : \mathbb{R}^m \to \mathbb{R}$ using a set of samples $S_M = \{x_n, f(x_n)\}_{n=1}^M$. A loss function could be defined by $\mathcal{L}(f_\theta, S_M) := \sum_{n=1}^M \|f_\theta(x_n) - f_n\|_2$, which enables formulating the following optimization problem over the parameters θ

$$\min_\theta \sum_{n=1}^{M} \|f_\theta(x_n) - f_n\|_2.$$

This optimization problem could be solved by using gradient descent or a similar iterative optimization algorithm. In its simplest form and assuming that the parameters are initialized by θ_0, the gradient descent iteration is given by

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta \mathcal{L}(f_\theta, S_M), \tag{3.1}$$

where $\eta > 0$ denotes the step size, often called the learning rate, and $\nabla_\theta \mathcal{L}(f_\theta, S_M x)$ is the gradient of the loss function with respect to the set of parameters θ. A slightly more sophisticated variation of gradient descent is the algorithm called Adam, introduced by Kingma and Ba (2014). Here, the step size of each iteration, η_i, is adaptive, and the new gradient direction considers the gradient in step i as well as the previous step $i - 1$.

On the other hand, optimization methods based on the root finding algorithm Newton–Raphson, could be shown to have quadratic convergence under some regularity conditions (Nocedal and Wright, 1999, Ch. 7). These methods do however require the computation of the inverse Hessian $H := (\nabla_\theta^2 \mathcal{L})^{-1}$. However, the aim of quasi-Newton methods is to derive algorithms simplifying this calculation. The limited-memory Broyden, Fletcher, Goldfarb and Shannon (L-BFGS) algorithm is described by (Nocedal and Wright, 1999, Ch. 7.2) and is a scheme for iteratively updating the inverse Hessian approximations using curvature information from earlier iterations, in a memory efficient way.

Studying the deep learning literature, the Adam algorithm is often the default choice when selecting an optimization procedure. However, as shown by Le et al. (2011), the L-BFGS algorithm is in many cases superior, in particular when the domain of the neural network F_θ is low dimensional, and the number of parameters is not too large. In the setting of Hamiltonian neural networks studied in this thesis, the problems are typically of low dimensionality and the number of parameters is relatively low. Furthermore, during the implementation of the numerical experiments, we experienced that L-BFGS had better performance than Adam. Hence, L-BFGS will be used as optimization procedure in this setting.

3.2 Scientific works on ODEs and deep learning

Deep learning and neural networks have typically been studied and developed within the realm of computer science, whereas differential equations is one of the major areas studied in mathematics. The combination of deep learning and differential equations has in the recent years generated substantial research activity. This research is oriented in some major directions:

1. Using ODEs to describe the optimization iterations as a gradient flow.

2. Interpreting neural network layers as the discretization of an ODE.

3. Using neural networks to approximate terms of an ODE.

4. Approximating functions in energy based equations from classical mechanics.

3.2.1 Gradient descent as continuous gradient flows

The first direction studies the continuous version of the gradient descent step in Equation (3.1), which could be rewritten by

$$\frac{\theta_{i+1} - \theta_i}{\eta} = \nabla_\theta \mathcal{L}(f_\theta, S_M).$$

This could be interpreted as an explicit Euler discretization of an ODE with step size η. The corresponding continuous system of $\theta(t)$, where $\theta_{i+1} \approx \theta(t+\eta)$, could be found by taking the limit such that $\eta \to 0$. We get an ODE or a *gradient flow* by

$$\frac{d}{dt}\theta(t) = \nabla_\theta \mathcal{L}(f_{\theta(t)}, S_M).$$

In particular, the authors of Elkabetz and Cohen (2021) argue that the theoretical analysis of the continuous gradient flow is able to generate results that could be used to understand the gradient descent iterations for simple neural networks better. On the other hand, Shi et al. (2019) consider the different optimization methods obtained when different numerical integrators are used to discretize the gradient flow.

3.2.2 ResNet layers as an ODE discretization

The residual neural network (ResNet) is a modified version of the multilayer perceptron by Definition 3.1. Here, each layer is a parametrized perturbation of the identity map, such that one layer is given by

$$a_{i+1} = a_i + h\sigma\Big(W_i a_i + b_i\Big).$$

This could be interpreted as an explicit Euler discretization of an ODE with solution $a(t)$, where the step size is given by $h > 0$. Letting $h \to 0$, the continuous limit to the ResNet is given by

$$\dot{a}(t) = \sigma\Big(W(t)a(t) + b(t)\Big).$$

In this setting, Haber and Ruthotto (2017) and Weinan (2017) use tools from the field of dynamical systems to develop strategies to stabilize the learning procedure of deep ResNets. This could be achieved by restricting the weight matrices W_i to be antisymmetric, or by recasting the forward propagation as a Hamiltonian system and discretizing the arising system using symplectic integrators. Building on this, Chen et al. (2018) suggest formulating neural network dynamics more generally by the ODE

$$\dot{y}(t) = f_\theta(t, y(t)),$$

or a neural ODE, following the terminology of the authors. Here, f_θ describes the dynamics of the system and could take different forms, as long as the activation functions are continuously differentiable and Lipschitz in order to guarantee uniqueness of the solution $y(t)$. Chen et al. (2018) suggests formulating an adjoint ODE system in order to find the gradients of θ, which has to be solved in reverse time. This approach first derives gradients for solving the optimization problem, before discretization. The first-optimize-then-discretize approach is studied more thoroughly by Benning et al. (2019). Here, the neural network training problem is formulated as an optimal control problem aiming at minimizing some loss function where a neural network defines the vector field of an ODE constraint. Furthermore, the properties of the systems obtained when applying different Runge–Kutta methods to discretize both the ODE constraint and the adjoint ODE, are studied. Discretizing using implicit integration schemes and the issue of computing parameter gradients in this case, is studied by Bai et al. (2019) and several other recent papers.

3.2.3 Approximating ODE terms using neural networks

Differential equations are widely used in a range of scientific disciplines. However, in many cases it is challenging to identify a specific analytical formulation of the dynamics of a complicated system. The hybrid approach obtained when merging deep learning with the tools of traditional mathematical modelling has gained traction in the recent years. Raissi, Perdikaris and Karniadakis introduced the concept of physics informed neural networks Raissi et al. (2019). Here, the solution of an ODE or a partial differential equation (PDE) is replaced by a neural network $u_\theta(x,t)$ with $x \in \Omega \subset \mathbb{R}^n$ and $t \in [0,T]$. A PDE could thus be formulated as

$$\frac{d}{dt}u_\theta + \mathcal{N}[u_\theta] = 0,$$

where $\mathcal{N}[\ \cdot\ ]$ is a general differential operator. This equation defines a target that could be used for training the network u_θ where boundary and initial conditions also would have to be included to ensure uniqueness of a solution. Proposing a more general framework, Rackauckas et al. (2020) allow neural networks to represent unknown terms in a wide range of mathematical models of dynamical systems. Implementing general algorithms for computing the adjoint allows for compatibility with a range of different numerical methods. A somewhat different approach was introduced by Poli et al. (2020), which use a neural network to approximate the local error term of a numerical one-step integration method. Considering a step with the first order explicit Euler method over a known vector field f, this approach would be equivalent to

$$y_{n+1} = y_n + hf(y_n) + h^2 g_\theta(y_n),$$

where g_θ could be trained on data obtained by using a higher order integrator.

3.2.4 Approximating functions in energy based ODEs

Classical mechanics develops a theory on how the dynamics of a physical system could be described by equations based on the kinetic and potential energy of the system. As briefly introduced in section 2.3, the Hamiltonian equations could be numerically integrated to fully describe the dynamics of a system. This setting is the starting point for Hamiltonian neural networks and a series of recent scientific works within the realm of modelling physical systems based on the Lagrangian or Hamiltonian formulation.

A Hamiltonian system, where $y = [q, p]^T$ and $q, p \in \mathbb{R}^d$ are defined as the canonical coordinates, is defined by

$$\dot{y} = J\nabla H(y).$$

The Hamiltonian neural network was introduced by Greydanus et al. (2019) and propose to learn the dynamics of a Hamiltonian system, as seen in Equation (2.24), by parametrizing the Hamiltonian as a neural network by $H_\theta(y)$. It could then be trained either on observations of both $\{\dot{y}(t_i), y(t_i)\}_{i=0}^n$ or if only $y(t_i)$ is available, approximate the time derivative by finite differences.

In addition to preserving the Hamiltonian, the exact flow of Hamiltonian ODEs is a symplectic transformation, which was seen in Section 2.3. Zhu et al. (2020) use backwards error analysis to show that symplectic integrators guarantee the existence of a modified Hamiltonian which is the learning target for the neural network. David and Méhats (2021) build on this by deriving a correction term perturbing the modified Hamiltonian such that it more accurately represents the underlying Hamiltonian of the observed system. A more detailed analysis of how a symplectic integrator modifies the learning target and how the integration error is cancelled when using the same integrator for training and integration after training, is provided by Offen and Ober-Blöbaum (2022). As already mentioned in Section 2.7, Matsubara et al. (2019) derive an algorithm to compute discrete gradients of neural networks, allowing the learned Hamiltonian to be preserved exactly during integration. Celledoni et al. (2022) present a more rigorous approach on how to deal with the geometry of constrained mechanical system. Here, problems with solutions evolving on a Lie group are studied and Lie-group integrators preserving the manifold structure are compared to classical Runge–Kutta methods.

The equations of Hamiltonian systems come with the assumption of energy conservation. Their generalization into port-Hamiltonian systems is obtained by including dissipative terms and allowing forcing- or control-terms in the equation. This enables the derivation of a unified equation describing the interconnection of multiple physical systems as described by Van Der Schaft and Jeltsema (2014). Zhong et al. (2020) and Duong and Atanasov (2021) use the port-Hamiltonian formulation to approximate the underlying Hamiltonian, potential forcing terms and terms describing dissipation. Eidnes et al. (2022) offer a demonstration of the flexibility of the port-Hamiltonian formalism, using it to model a system of multiple connected chemical tanks with interacting forces.

On the other hand, Chen et al. (2019) assume that the Hamiltonian is separable and use the Störmer–Verlet method for discretization. This is a second order, symplectic method which is explicit for separable Hamiltonian systems. Finzi et al. (2020) assume that the kinetic energy-term could be modelled as a quadratic form where the mass matrix could be learned directly.

Finally, an interesting approach is to construct neural networks to have similar mathematical properties as the flow map of the ODE it is approximating. Since the flow map of a Hamiltonian system is symplectic, Jin et al. (2020) parametrize symplectic transformations that could be combined to form a trainable neural network that is a symplectic transformation by design. Several other works explore a similar approach, such as Chen and Tao (2021) and Chen et al. (2021).

Chapter 4

Integration methods for inverse ODE problems

The majority of the approaches to learning Hamiltonian systems from observations use numerical integrators in the training. However, there has not been much work on systematically exploring the space of different types of integrators and which properties are beneficial in the setting of inverse problems on Hamiltonian ODEs. Here, we show that a particular class of Runge–Kutta methods, namely the mono-implicit Runge–Kutta methods (MIRK), are computationally efficient for inverse ODE problems. However, as proved below, this class does not contain symplectic integrators of order higher than $p = 2$. We show how a recently introduced MIRK method, suggested by Eidnes et al. (2022), can be derived by modifying the Gauss collocation method of order $p = 4$. Furthermore, the mean inverse integrator is introduced and analyzed on how it propagates noise in comparison to a one-step method.

4.1 Inverse ODE problems

Let a first order differential equation be defined by a vector field $f : \mathbb{R}^m \to \mathbb{R}^m$ such that for a given initial value $y_0 \in \mathbb{R}^m$ a solution $y(t)$ has to satisfy

$$\dot{y} = f(y). \tag{4.1}$$

The inverse ODE problem assumes that the vector field f is unknown, but that observations, or approximations to the exact flow, are available for M different initial values, at $N + 1$ different times given by

$$y_i^j \approx \varphi_{ih,f}(y_0^j) \quad \text{with} \quad y_0 \in S_0 \subset \mathbb{R}^m$$

for $i = 0, \ldots, N$ and $j = 1, \ldots, M$, assuming uniform time steps $t_i = ih$ and letting the initial values be contained in

$$S_0 = \{y_0 \text{ s.t. } \|y_0\|_2 \leq r\}, \tag{4.2}$$

for some $r > 0$. Let us denote the set of observed, discretized solutions by

$$S_N^M = \left\{y_i^j\right\}_{i=0,j=1}^{N,M}.$$

For simplicity, it is sometimes convenient to consider a discretized flow from on only a single initial value. In that case, the set of observations will be denoted by $S_N = \{y_i\}_{i=0}^N$. An inverse ODE problem could then be defined by:

Definition 4.1 (Inverse ODE problem)
Let f_θ be a neural network with parameters θ, Φ_{h,f_θ} be a one-step integration method with step length $h > 0$ and S_N^M be a set of samples from the flow of an ODE with vector field $f(y)$. The inverse problem is the following optimization problem

$$\begin{aligned} &\min_\theta \mathcal{L}(\Phi_{h,f_\theta}, S_N^M, f_\theta) \\ \textit{where} \quad &\mathcal{L}(\Phi_{h,f_\theta}, S_N^M, f_\theta) := \sum_{m=1}^{M} \sum_{n=0}^{N-1} \left\| y_{n+1}^m - \Phi_{h,f_\theta}(y_n^m) \right\|. \end{aligned}$$

Remark
In this thesis we assume that all ODEs are Hamiltonian systems and follow the idea of Greydanus et al. (2019) of learning the Hamiltonian instead of the full vector field. We will thus train a neural network on the form

$$f_\theta(y) := J\nabla H_\theta(y),$$

such that the learned vector field will always form a Hamiltonian system.

4.2 Inverse explicit integration methods

The inverse ODE problem requires rethinking how numerical integrators are used, since we assume that multiple, sequential points in the discretized flow are provided by the sample S_N^M. Hence, integrators which are implicit when considering the classical initial value problem could be made explicit. This is exploited in several works on Hamiltonian neural networks such as David and Méhats (2021), Celledoni et al. (2022), Eidnes et al. (2022) and more. Here, this difference is made clear by introducing the simple concept of the inverse injection.

Definition 4.2 (Inverse injection)
Let $S_N = \{y_i\}_{i=0}^N$ be observations from an ODE by Equation (4.1) *and $\Phi_{h,f}$ be an implicit integration scheme such that integration requires the solution of a system over $\hat{y}_{n+1}$, by*

$$\hat{y}_{n+1} = \Phi_{h,f}(y_n, \hat{y}_{n+1}), \tag{4.3}$$

with $y_n \in S_N$. The inverse injection consists in replacing $\hat{y}_{n+1}$ in Equation 4.3 with the observed value of the solution $y_{n+1} \in S_N$, yielding a method given by

$$\hat{y}_{n+1} = \Phi_{h,f}(y_n, y_{n+1}).$$

The notation $\Phi_{h,f}(y_n, y_{n+1})$ highlights the fact that taking the integration step (with an implicit integrator) depends on both y_n and y_{n+1}. Furthermore, it is necessary to make a distinction between integrators that are explicit and integrators that are not, when using the inverse injection.

Definition 4.3 (Inverse explicit)
Let $\Phi_{h,f}$ be a numerical one-step method. $\Phi_{h,f}$ is inverse explicit if the integration scheme is explicit under the inverse injection.

This concept is useful, as inverse explicit integrators, in the same manner as explicit integrators for the forward problem, mitigate the need to solve a system of equations before taking a step. It is thus useful to investigate which integrators belong to this class.

Lemma 4.4
Mono-implicit Runge–Kutta methods are inverse explicit.

Proof. For Runge–Kutta methods we have that

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i.$$

As the stages of the MIRK methods could be written on the form given by Equation (2.15) and by the fact that $[D]_{ij}$ is strictly lower triangular, it is clear that

$$\begin{aligned}
k_1 &= f\Big(y_n + v_1(y_{n+1} - y_n)\Big) \\
k_2 &= f\Big(y_n + v_2(y_{n+1} - y_n) + hd_{21}k_1)\Big) \\
&\vdots \\
k_s &= f\Big(y_n + v_s(y_{n+1} - y_n) + h\sum_{j=1}^{s-1} d_i k_j\Big).
\end{aligned}$$

Hence, each stage could be computed without solving any system of equations, when it is assumed that y_n and y_{n+1} is known. This means that the stages and the method is explicit under the inverse injection. □

Lemma 4.5
The discrete gradient integration methods are inverse explicit.

Proof. The methods depend only on y_n, y_{n+1} and have no implicit intermediate stages. It is therefore straightforward to verify that they are explicit under the inverse injection.
□

It is evident that the MIRK methods studied by Burrage et al. (1994) could be very useful for solving inverse ODE problems, together with the discrete gradient methods including the ones to an arbitrary order, by Eidnes (2022). However, in the setting of Hamiltonian ODEs, we have seen that symplecticity is a useful property. Therefore, it is of relevance to explore the maximum order of symplectic MIRK methods.

Theorem 4.6
The maximum order of a symplectic MIRK method is $p = 2$.

Proof. As shown in Section 2.4.2, a Runge–Kutta method is symplectic if and only if

$$b_i a_{ij} + b_j a_{ji} - b_i b_j = 0.$$

Inserting the MIRK coefficients a_{ij} by Definition 2.11, we get

$$\begin{aligned} b_i(v_i b_j + d_{ij}) + b_j(v_j b_i + d_{ji}) - b_i b_j &= 0 \\ b_i d_{ij} + b_j d_{ji} + b_i b_j (v_j + v_i - 1) &= 0. \end{aligned}$$

As D is strictly lower triangular, we get that

$$\begin{aligned} \text{either} \quad d_{ji} = 0 \quad \text{or} \quad d_{ij} = 0 \implies \quad & b_i d_{ij} + b_i b_j (v_j + v_i - 1) = 0 \\ \text{if } i = j \implies \quad & b_i^2 (2v_i - 1) = 0 \end{aligned}$$

Requiring d_{ij}, b_i and v_i to satisfy the symplecticity condition, yields the following restriction

$$\begin{aligned} b_i d_{ij} + b_i b_j (v_j + v_i - 1) = 0, &\quad \text{for } i \neq j, \\ \text{and} \quad b_i = 0 \text{ or } v_i = \frac{1}{2}, &\quad \text{for } i = j. \end{aligned} \tag{4.4}$$

We get these conditions based on whether or not $b_i = 0$:

	$b_j = 0$	$b_j \neq 0$
$b_i = 0$	$d_{ij} \in \mathbb{R}$ $v_i, v_j \in \mathbb{R}$	$d_{ij} \in \mathbb{R}$ $v_i, v_j \in \mathbb{R}$
$b_i \neq 0$	$d_{ij} = 0$ $v_i, v_j \in \mathbb{R}$	$d_{ij} = 0$ $v_i, v_j = \frac{1}{2}$

Without loss of generality we assume that the m first entries of $b \in \mathbb{R}^s$ are zero. Enforcing the conditions of Equation (4.4) on $v \in \mathbb{R}^s$ we get

$$\begin{aligned} b &= [0, \dots, 0, b_{m+1}, \dots, b_s]^T, \\ v &= [v_1, \dots, v_m, \frac{1}{2}, \dots, \frac{1}{2}]^T. \end{aligned}$$

In total, the MIRK coefficient matrix $A = D + vb^T$ gives a Butcher tableau of the form

$$\begin{array}{c|ccccc|ccc}
 & 0 & 0 & 0 & \dots & 0 & v_1 b_{m+1} & \dots & v_1 b_s \\
 & d_{21} & 0 & 0 & \dots & 0 & & & \\
 & d_{31} & d_{32} & 0 & & 0 & \vdots & & \vdots \\
 & \vdots & & & \ddots & & & & \\
 & d_{m,1} & \dots & \dots & d_{m,m-1} & 0 & v_m b_{m+1} & \dots & v_m b_s \\
\hline
 & 0 & & \dots & & 0 & \frac{1}{2} b_{m+1} & \dots & \frac{1}{2} b_s \\
 & \vdots & & & & \vdots & \vdots & & \vdots \\
 & \vdots & & & & \vdots & \vdots & & \vdots \\
 & 0 & & \dots & & 0 & \frac{1}{2} b_{m+1} & \dots & \frac{1}{2} b_s \\
\hline
 & 0 & & \dots & & 0 & b_{m+1} & \dots & b_s
\end{array}$$

Since the lower left submatrix is the zero matrix, this leaves the stages $k_{m+1}, \dots, k_s$ unconnected to the first m stages. Furthermore as $b_i = 0$ for $i = 1, \dots, m$, these stages are not included in the computation of the final integration step. The method is thus reducible to the lower right submatrix of A and $b_{m+1}, \dots, b_s$. The reduced method is in general given by

$$\begin{array}{c|ccc}
 & \frac{1}{2} b_1 & \dots & \frac{1}{2} b_s \\
 & \vdots & & \vdots \\
 & \frac{1}{2} b_1 & \dots & \frac{1}{2} b_s \\
\hline
 & b_1 & \dots & b_s
\end{array}$$

It is trivial to check that, if $\sum_i^s b_i = 1$ the method satisfies order conditions up to order $p = 2$, which could be found in (Hairer et al., 2006, Ch. III.1.1) to be

$$\sum_i b_i = 1,$$
$$\sum_{i,j} b_i a_{ij} = \frac{1}{2},$$

but fails to satisfy the first of the two conditions required for order $p = 3$, since

$$\sum_{i,j,k} b_i a_{ij} a_{ik} = \frac{1}{4} \sum_{i,j,k} b_i b_j b_k = \frac{1}{4} \neq \frac{1}{3}.$$

Hence, the maximum order of a symplectic MIRK method is $p = 2$.

□

Solving Hamiltonian inverse ODE problems and requiring a MIRK method to be inverse explicit and symplectic, leaves us with a method of maximum order $p = 2$, where the implicit midpoint method is a prominent example. However, considering the class of partitioned Runge–Kutta methods, see for instance (Hairer et al., 2006, Ch. II.2), there are symplectic, inverse explicit methods of order $p > 2$. Consider for instance the method of order $p = 4$ by Yoshida (1990).

4.3 Symmetric MIRK methods

Consider the exact flow map of an autonomous ODE, $\varphi_{h,f}(y(t_0))$. Let $h = t_1 - t_0$ such that $y(t_1) = \varphi_{h,f}(y(t_0))$. The inverse map is by definition the map $\varphi^{-1}_{h,f}$, such that

$$\begin{aligned} \varphi^{-1}_{h,f} \circ \varphi_{h,f}(y(t_0)) &= y(t_0), \\ \iff \varphi^{-1}_{h,f}(y(t_1)) &= y(t_1) + \int_{t_1}^{t_0} f(y(t))\, dt = y(t_0). \end{aligned}$$

Hence, taking a negative time step $-h$ in the inverse flow, leaves us with

$$\begin{aligned} \varphi^{-1}_{-h,f}(y(t_0)) &= y(t_0) - \int_{t_1}^{t_0} f(y(t))\, dt \\ &= y(t_0) + \int_{t_0}^{t_1} f(y(t))\, dt \\ &= \varphi_{h,f}(y(t_0)) \end{aligned}$$

It is thus clear that the adjoint of the exact flow, meaning the inverse flow map with negative time step, is the same map. In other words, the exact flow is self-adjoint. As discussed in (Hairer et al., 2006, Ch. II.3) this property is not necessarily shared by a numerical one-step map. The adjoint numerical method is defined by:

Definition 4.7
Let $\Phi_{h,f}$ be a numerical one-step integrator. The adjoint $\Phi^*_{h,f}$ is given by the inverse of the original method with negative step size $-h$, such that

$$\Phi^*_{h,f} = \Phi^{-1}_{-h,f}\ .$$

An integrator satisfying $\Phi^*_{h,f} = \Phi_{h,f}$ is called a symmetric integrator. It also makes sense to call such an integrator self-adjoint. The symmetry property, similarly as symplecticity, could be shown to give rise to improved long time behavior of numerical integrators, as discussed in (Hairer et al., 2006, Ch. V) and (Hairer et al., 2006, Ch. XI). Hence, in this section two approaches to obtaining symmetric MIRK methods, will be considered.

4.3.1 Symmetric MIRK by modified Gauss collocation

Studying the Gauss collocation method with two stages and order $p = 4$ (GC4) found in Example 2.10, it is clear that this method is not mono-implicit. However, by introducing two new stages, the method could be modified to a new MIRK method which is also symmetric and of order $p = 4$. This method was first derived by Sølve Eidnes as an approximation of order $p = 4$ to the modified midpoint rule of Chartier et al. (2007) and presented by Eidnes et al. (2022). This new method is however, not symplectic.

GC4 is given by the following stages, where we let $c = \frac{\sqrt{3}}{6}$

$$\begin{aligned} k_1 &= f\left(y + h\big(\frac{k_1}{4} + (\frac{1}{4} - c)k_2\big)\right), \\ k_2 &= f\left(y + h\big(\frac{k_2}{4} + (\frac{1}{4} + c)k_1\big)\right), \\ \hat{y} &= y + \frac{h}{2}(k_1 + k_2). \end{aligned}$$

However, the stages could be rewritten, such that

$$\begin{aligned} k_1 &= f\left(y + h\big(\frac{k_1}{4} + (\frac{1}{4} - c)k_2\big)\right) \\ &= f\left(y + \frac{h}{4}\big(k_1 + k_2\big) - chk_2\right) \\ &= f\left(\frac{1}{2}(y + \hat{y}) - chk_2\right), \\ \text{and similarly} \quad k_2 &= f\left(\frac{1}{2}(y + \hat{y}) + chk_1\right). \end{aligned}$$

Thus, if we are able to approximate the chk_1 and chk_2 terms with symmetric, inverse explicit stages, whilst maintaining the order, we will have a fourth order, four stage MIRK method.

Theorem 4.8
Let $c = \frac{\sqrt{3}}{6}$. The Runge–Kutta method defined by the stages

$$\begin{aligned} \tilde{k}_1 &= f\left(y + \frac{h}{4}(1 - 2c)\big(\hat{k}_1 + \hat{k}_2\big)\right) \\ \tilde{k}_2 &= f\left(y + \frac{h}{4}(1 + 2c)\big(\hat{k}_1 + \hat{k}_2\big)\right) \\ \hat{k}_1 &= f\left(y + \frac{h}{4}(\hat{k}_1 + \hat{k}_2) - ch\tilde{k}_2\right) \\ \hat{k}_2 &= f\left(y + \frac{h}{4}(\hat{k}_1 + \hat{k}_2) + ch\tilde{k}_1\right) \\ \hat{y} &= y + \frac{h}{2}(\hat{k}_1 + \hat{k}_2) \end{aligned}$$

is symmetric, of order $p = 4$, and A-stable.

Proof. By Taylor expansion around y, and denoting $f := f(y)$ we find that for k_1 and

$\hat{k}_1$ we have

$$\begin{aligned}
k_1 &= f + h(\frac{1}{2} - c)f'f + \frac{h^2}{2}(\frac{1}{2} - c)^2 f''(f,f) + h^2(\frac{1}{4} - \frac{c}{2} - c^2)f'f'f + \mathcal{O}(h^3),\\
\hat{k}_1 &= f + h(\frac{1}{2} - c)f'f + \frac{h^2}{2}(\frac{1}{2} - c)^2 f''(f,f) + \frac{h^2}{4}f'f'f - \frac{ch^2}{2}(1+2c)f'f'f + \mathcal{O}(h^3)\\
&= f + h(\frac{1}{2} - c)f'f + \frac{h^2}{2}(\frac{1}{2} - c)^2 f''(f,f) + h^2(\frac{1}{4} - \frac{c}{2} - c^2)f'f'f + \mathcal{O}(h^3),
\end{aligned}$$

and similarly for k_2 and $\hat{k}_2$ that

$$\begin{aligned}
k_2 &= f + h(\frac{1}{2} + c)f'f + \frac{h^2}{2}(\frac{1}{2} + c)^2 f''(f,f) + h^2(\frac{1}{4} + \frac{c}{2} - c^2)f'f'f + \mathcal{O}(h^3),\\
\hat{k}_2 &= f + h(\frac{1}{2} - c)f'f + \frac{h^2}{2}(\frac{1}{2} - c)^2 f''(f,f) + \frac{h^2}{4}f'f'f + \frac{ch^2}{2}(1-2c)f'f'f + \mathcal{O}(h^3)\\
&= f + h(\frac{1}{2} - c)f'f + \frac{h^2}{2}(\frac{1}{2} - c)^2 f''(f,f) + h^2(\frac{1}{4} + \frac{c}{2} - c^2)f'f'f + \mathcal{O}(h^3),
\end{aligned}$$

Hence, it is clear that

$$\begin{aligned}
\hat{k}_1 &= k_1 + \mathcal{O}(h^3),\\
\hat{k}_2 &= k_2 + \mathcal{O}(h^3),
\end{aligned}$$

meaning that the for the step $\hat{y}$, the MIRK method is an approximation of the GC4 method with an error of $\mathcal{O}(h^4)$ and has at least order $p = 3$. However, by introducing $\overline{y} := \frac{\hat{y}+y}{2}$ the MIRK method could be written on the form

$$\begin{aligned}
\tilde{k}_1 &= f\Big(\overline{y} - c\big(\hat{y} - y\big)\Big),\\
\tilde{k}_2 &= f\Big(\overline{y} + c\big(\hat{y} - y\big)\Big),\\
\hat{k}_1 &= f\Big(\overline{y} - chf\Big(\overline{y} + c\big(\hat{y} - y\big)\Big)\Big),\\
\hat{k}_2 &= f\Big(\overline{y} + chf\Big(\overline{y} - c\big(\hat{y} - y\big)\Big)\Big).
\end{aligned}$$

Denoting the adjoint stages and steps by k_i^* and y^*, and finding them by interchanging $\hat{y}$ and y in addition to taking negative step size $-h$, it is trivial to see from above, that $\hat{k}_1^* = \hat{k}_2$ and $\hat{k}_2^* = \hat{k}_1$. We thus find the adjoint step $\hat{y}^*$ by

$$\begin{aligned}
y &= \hat{y}^* - \frac{h}{2}(\hat{k}_1^* + \hat{k}_2^*)\\
\implies \hat{y}^* &= y + \frac{h}{2}(\hat{k}_2 + \hat{k}_1)
\end{aligned}$$

The adjoint method is the same as the original method and is thus symmetric. By Theorem 3.2 in (Hairer et al., 2006, Ch. II.3), a symmetric method has even order.

Since the MIRK method has at least order $p = 3$ and is symmetric, it thus follows that it is of (at least) order $p = 4$.

The linear stability function (Wanner and Hairer, 1996, Ch. IV.3) could be used to characterize how well the solver handles stiff equations. The stability function for a Runge–Kutta method, is in general given by

$$R(z) = \frac{\det\Big(I - zA + z\mathbb{1}b^T\Big)}{\det\Big(I - zA\Big)},$$

where $\mathbb{1} := [1, \dots, 1]^T \in \mathbb{R}^s$. By computing the determinants for the MIRK method, this can be shown to be given by

$$R(z) = \frac{\frac{z^2}{12} + \frac{z}{2} + 1}{\frac{z^2}{12} - \frac{z}{2} + 1},$$

which is identical to the stability function of GC4 called Hammer–Hollingsworth order 4 in (Wanner and Hairer, 1996, Ch. IV.3). GC4 is A-stable following (Hairer et al., 2006, Ch. II.1.3), and it follows that this MIRK method is A-stable. $\square$

The classical Butcher-tableau (left), and the extended MIRK tableau (right) for this MIRK method is given by

$$\begin{array}{c|cccc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & 0 & 0 & \frac{1}{4} - \frac{\sqrt{3}}{12} & \frac{1}{4} - \frac{\sqrt{3}}{12} \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & 0 & 0 & \frac{1}{4} + \frac{\sqrt{3}}{12} & \frac{1}{4} + \frac{\sqrt{3}}{12} \\
\frac{1}{2} - \frac{\sqrt{3}}{6} & 0 & -\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{\sqrt{3}}{6} & 0 & \frac{1}{4} & \frac{1}{4} \\
\hline
 & 0 & 0 & \frac{1}{2} & \frac{1}{2}
\end{array}
\qquad
\begin{array}{c|c|cccc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{2} - \frac{\sqrt{3}}{6} & 0 & 0 & 0 & 0 \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{2} + \frac{\sqrt{3}}{6} & 0 & 0 & 0 & 0 \\
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{2} & 0 & -\frac{\sqrt{3}}{6} & 0 & 0 \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{2} & \frac{\sqrt{3}}{6} & 0 & 0 & 0 \\
\hline
 & & 0 & 0 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

4.3.2 Symmetric MIRK by the adjoint

One systematic approach to obtaining symmetric inverse explicit Runge–Kutta methods involves the adjoint of a method, which could be found by Definition 4.7. This approach results in identical methods to what is called ATPERK methods by Enright and Muir (1986).

As explained in the proof of Theorem 4.8, the adjoint method could be found by interchanging y_{n+1} and y_n in addition to changing the sign of the step size h. For a Runge–Kutta method, denoting the vector field evaluations by

$$\hat{F}(y_n, y_{n+1}, h) := \sum_{i=1}^{s} b_i k_i,$$

the method and its adjoint could thus be given by

$$\begin{aligned}
&\Phi_{h,f}(y_n) = y_n + h\hat{F}(y_{n+1}, y_n, h) \\
\implies\ &\Phi_{h,f}(y_n)^* = y_n + h\hat{F}(y_n, y_{n+1}, -h),
\end{aligned}$$

such that a symmetric method could be constructed by taking the mean of the original and adjoint vector field evaluation. The following is a well known result that we state here formally.

Theorem 4.9
Let a Runge–Kutta method be given by $\Phi_{h,f}(y_n)$. The method

$$\Psi_{h,f}(y_n) = \frac{1}{2}\Big(\Phi_{h,f}(y_n) + \Phi^*_{h,f}(y_n)\Big),$$

is symmetric and of even order.

Proof. The adjoint of $\Psi_{h,f}(y_n)$ is given by

$$\Psi^*_{h,f}(y_n) = \frac{1}{2}\Big(\Phi^*_{h,f}(y_n) + \Phi_{h,f}(y_n)\Big),$$

since $(\Phi^*_{h,f})^* = \Phi_{h,f}$. Thus $\Psi^*_{h,f} = \Psi_{h,f}$, so $\Psi_{h,f}$ is symmetric, or self-adjoint. By Theorem 3.2 in (Hairer et al., 2006, Ch. II.3), the method has even order. □

The implicit trapezoidal rule could be shown to be a method obtained by taking the mean of a step with explicit Euler and the adjoint method, which is implicit Euler.

For a MIRK method defined by the coefficients $v \in \mathbb{R}^s$, $D \in \mathbb{R}^{s\times s}$ and $b \in \mathbb{R}^s$ we can find the adjoint of the stages by

$$\begin{aligned} k_i^* &= f\Big(y_{n+1} + v_i(y_n - y_{n+1}) - h\sum_{j=1}^{s} d_{ij}k_j\Big) \\ &= f\Big(y_n + (1-v_i)(y_{n+1} - y_n) + h\sum_{j=1}^{s} -d_{ij}k_j\Big), \end{aligned}$$

such that the original (left) and corresponding Butcher tableau of the adjoint MIRK method (right) is given by

$$\begin{array}{c|c|c} c & v & D \\ \hline & & b^T \end{array} \qquad \begin{array}{c|c|c} \tilde{c} & \mathbb{1} - v & -D \\ \hline & & b^T \end{array}$$

Taking the mean of the original MIRK method and the adjoint yields

$$\Psi_{h,f}(y_n) = y_n + \frac{h}{2}\sum_{i=1}^{s} b_i(k_i + k_i^*),$$

which is a Runge–Kutta method with $2s$ stages on the form

$$\begin{array}{c|cc} & vb^T + D & \mathbf{0} \\ & \mathbf{0} & (\mathbb{1} - v)b^T - D \\ \hline & \frac{b^T}{2} & \frac{b^T}{2} \end{array}$$

It is not difficult to see that the method stated here, is indeed the same as the ATPERK method derived in the context of a two-point boundary value problem, by Enright and Muir (1986). Here the authors conduct a more detailed analysis on computational efficiency and stability properties.

Example 4.10 (Two symmetric, inverse explicit Runge–Kutta methods)
The $s = 4$ stage method obtained from symmetrizing the third order MIRK method given in Example 2.13, is given by the Butcher tableau

$$
\begin{array}{c|cccc}
 & 0 & 0 & 0 & 0 \\
 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\
 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\
 & 0 & 0 & -\frac{1}{12} & \frac{5}{12} \\
\hline
 & \frac{1}{8} & \frac{3}{8} & \frac{1}{8} & \frac{3}{8}
\end{array}
$$

A MIRK method of order $p = 5$ is found by taking $c_2 = 0$ and $c_3 = \frac{4}{3}$ of the four stage method by Burrage et al. (1994). Symmetrizing this method yields a symmetric $s = 8$ stage method of order $p = 6$, but it has two zero rows in the A matrix, such that it could be reduced to the following method with $s = 7$ stages:

$$
\begin{array}{c|ccccccc}
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & \frac{73}{528} & \frac{43}{168} & -\frac{27}{1072} & \frac{78125}{123816} & 0 & 0 & 0 \\
 & \frac{205}{891} & \frac{422}{567} & -\frac{1}{67} & \frac{156250}{417879} & 0 & 0 & 0 \\
 & \frac{129163}{750000} & -\frac{194557}{2625000} & \frac{384021}{16750000} & \frac{89759}{281400} & 0 & 0 & 0 \\
 & \frac{43}{168} & 0 & 0 & 0 & \frac{73}{528} & -\frac{27}{1072} & \frac{78125}{123816} \\
 & -\frac{2215}{4536} & 0 & 0 & 0 & -\frac{119}{1296} & -\frac{11}{1072} & \frac{78125}{303912} \\
 & \frac{15472}{46875} & 0 & 0 & 0 & -\frac{35021}{1031250} & -\frac{100737}{2093750} & \frac{17246}{55275} \\
\hline
 & \frac{1457}{7392} & \frac{43}{336} & -\frac{27}{2144} & \frac{78125}{247632} & \frac{73}{1056} & -\frac{27}{2144} & \frac{78125}{247632}
\end{array}
$$

These methods could be further optimized by symmetrizing the methods before determining the free parameters (c_2 and c_3 for the $s = 4$ MIRK methods), and then finding the parameters that yield optimal stability properties.

4.4 Solving implicit equations in training

In Section 4.2 inverse explicit integrators were introduced as a special class of integration methods in the inverse ODE setting. The benefit of such methods are avoiding to solve non-linear systems of equations when taking an integration step. Using an implicit integration method in the neural network training, would require backpropagating gradients through every computation used for solving the equation. However, doing so would open the possibility of using higher order symplectic integrators such as the Gauss collocation methods in the training. As discussed in Section 2.4.2, symplecticity is a beneficial property when integrating Hamiltonian systems and higher integration order would increase accuracy.

For solving the systems, which was reformulated to a more efficient form in Section 2.6, there are multiple options such as using fixed-point iterations, the Newton–Raphson method or quasi-Newton methods avoiding computing the full Jacobian matrix at every iteration. Consider in general the problem of finding a fixed point $x^* \in \mathbb{R}^m$ such that for a given function $G : \mathbb{R}^m \to \mathbb{R}^m$ we have that

$$G(x^*) = x^*.$$

It is possible to prove that fixed point iterations converge, consider for instance Quarteroni et al. (2010), with some regularity assumptions on the domain $D \subset \mathbb{R}^m$, and assuming that G is contractive, or in other words, that it has a Lipschitz constant $C < 1$ for all $x \in D$. Then for any $x_0 \in D$, the iterations

$$x_{k+1} = G(x_k),$$

converges to the fixed point x^* at a rate given by

$$\|x_k - x^*\| \leq \frac{C^k}{1-C}\|x_1 - x_0\|.$$

On the other hand, Newton–Raphson exploits the derivatives, or the Jacobian of a system in order to find the root. Hence, we rewrite the fixed-point problem in Equation (4.4) to the problem of finding x^* such that

$$K(x^*) := G(x^*) - x^* = 0.$$

Let J_K denote the Jacobian matrix such that $[J_K]_{ij} := \frac{\partial K_i}{\partial x_j}$, then the Newton–Raphson iterations are given either by the inverse Jacobian or by solving the linear system

$$\begin{aligned} x_{k+1} &= x_k - J_K(x_k)^{-1} K(x_k), \\ \iff J_K(x_k)\Big(x_{k+1} - x_k\Big) &= K(x_k). \end{aligned}$$

For this method, it is possible to prove convergence Nocedal and Wright (1999), if the fixed-point x^* exists, that $J_K(x^*)$ is non-singular around x^* and that the initial guess x_0 is sufficiently close to x^*. In that case, we have quadratic convergence by

$$\|x^* - x_{k+1}\| \leq \tilde{C}\|x^* - x_k\|^2,$$

for some positive constant $\tilde{C}$. In the setting of numerical integration using implicit Runge–Kutta methods, fixed-point iterations are said to be a tool from the stone-age transforming the algorithm into an explicit method destroying stability properties, according to (Wanner and Hairer, 1996, Ch. IV.8). However, it should be noted that this considers the numerical solution of stiff ODEs. On the other hand, in the setting of geometric numerical integration of non-stiff problems, in (Sanz-Serna and Calvo, 2018, Ch. 5.5), several numerical experiments are conducted, finding fixed-point iterations to be more computationally efficient and in (Hairer et al., 2006, Ch. VIII.6.2) one can find the same conclusion.

The problem of solving non-linear systems involving neural network evaluations, is discussed by Bai et al. (2019). Here, differentiation through the solution of the fixed point equations are done by implicit differentiation, in order to obtain a more efficient backwards pass. In this setting, however, we will rely on using the basic fixed point iterations and naively perform backpropagation over all the iterations. The motivation is in part the results by Hairer et al. (2006) and Sanz-Serna and Calvo (2018), but also the simplicity of its implementation, in comparison to the Newton–Raphson method or quasi-Newton methods. However, it would be an interesting area of future research to compare different algorithms for solving the equations, and different implementations of the backwards pass. One such approach is studied by Bai et al. (2019).

Considering first general implicit Runge–Kutta methods, the fixed point the equation (2.21) could be solved by the following iterations

$$\begin{aligned} Z_i^{(0)} &= hc_i f(y_n), \\ Z^{(k+1)} &= h(A \otimes I_m)F(Z^{(k)}). \end{aligned} \tag{4.5}$$

Whereas solving the Equation (2.22) for MIRK methods could be done by the following iterations

$$\begin{aligned} z_n^{(0)} &= hf(y_n), \\ z_n^{(k+1)} &= h\hat{F}(y_n, z_n^{(k)} + y_n). \end{aligned} \tag{4.6}$$

The iterations are terminated when $\|Z^{(k+1)} - Z^{(k)}\| \leq \text{TOL}$ for the general Runge–Kutta methods and $\|z_n^{(k+1)} - z_n^{(k)}\| \leq \text{TOL}$ for the MIRK methods. In both cases, the iterations are also terminated when the number of iterations reaches a maximum limit $n_{\text{max iter}}$.

4.5 Structure of integration in training

In the inverse ODE problem, one assumes that one or multiple trajectories of solutions $y_i \approx \varphi_{ih,f}(y_0)$ for $i = 0, \dots, N$, are available. Considering the optimization problem in Definition 4.1, this represents one approach for how to structure the numerical integration for solving the inverse problem. However, there are different approaches to how this could be done. First, following Definition 4.1, one-step explicit integration with the inverse injection

$$\hat{y}_{n+1} = \Phi_{h,f_\theta}(y_n, y_{n+1}) \quad n = 0, \dots, N,$$

secondly one-step, with an implicit integrator

$$\hat{y}_{n+1} = \Phi_{h,f_\theta}(y_n, \hat{y}_{n+1}) \quad n = 0, \dots, N,$$

or third, multiple implicit steps from one initial value y_0

$$\begin{aligned} \hat{y}_1 &= \Phi_{h,f_\theta}(y_0, \hat{y}_1), \\ \hat{y}_{n+1} &= \Phi_{h,f_\theta}(\hat{y}_n, \hat{y}_{n+1}) \quad n = 1, \dots, N. \end{aligned}$$

Let us denote the first approach as explicit one-step training, the second as implicit one-step training and the third approach as implicit multistep training. The difference between the dependency of the sequential observations y_i is made clear by Figure 4.1.

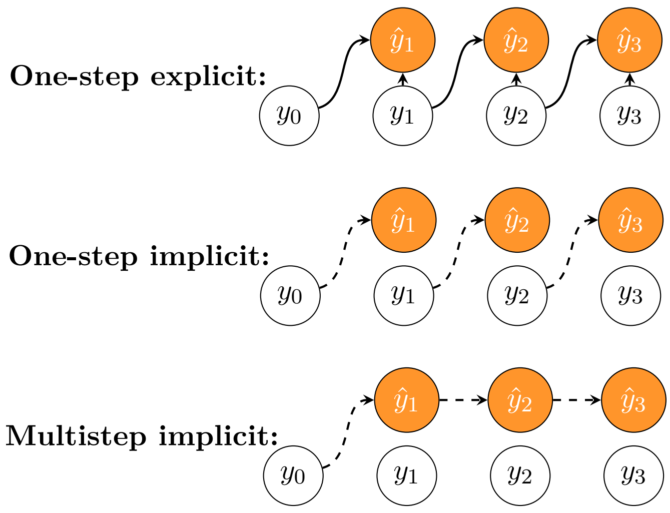


Figure 4.1: Illustration of the differences of the observation dependency, assuming $N = 3$ for explicit one-step training, implicit one-step training and implicit multistep training.

As mentioned in Section 4.2, the inverse injection yielding the one-step explicit approach, is used in several other works. It is also interesting to note that this approach is somewhat similar to teacher forcing Williams and Zipser (1989), used in the training of recurrent neural networks. Implicit multistep training is similar to what is done by Chen et al. (2019), however in this setting the ODEs studied are assumed to be separable Hamiltonian systems such that the second order, symplectic Störmer–Verlet integrator could be given on an explicit form. Here, the authors find that the multistep (or recurrent, following their terminology), for some test problems, has stronger performance when training on data with noise. Assuming noisy data, they also perform initial state optimization by simultaneously optimizing over the neural network parameters and searching for an initial value y_0 that minimizes the loss.

Introducing iterative schemes to solve the equations (2.21), (2.22) given by implicit integration steps in Chapter 2.6 allows for both using symplectic integrators of higher order $p > 2$, such as Gauss collocation methods of stages $s = 3, 4$, and to perform implicit multistep training on Hamiltonian systems that are not necessarily separable.

4.6 Integrators for noisy inverse problems

Assume we are solving an inverse problem with observations from a system that could be described by an underlying ODE. It is often necessary to assume that the observations are not exact, but perturbed by noise. This could be due to inexact measurements, external, chaotic forces perturbing the system being measured or a range of other different factors depending on the specific setting. In this section, we will derive an alternative formulation of the optimization problem found in Definition 4.1 by rethinking how to utilize numerical one-step methods. It leverages the algebraic structure of the exact flow map to get an optimization target that is less sensitive to noise.

4.6.1 Noisy ODE observations

Here, the noise of the observations will be modelled assuming it is independent and identically normally distributed.

Definition 4.11 (Noisy ODE observation)
Let $S_N^M = \{y_i^j\}_{i=0,j=1}^{N,M}$ be observations from an ODE for a vector field $f(y)$. A noisy ODE observation is defined by the perturbation

$$\tilde{y}_i^j = y_i^j + \delta_{ij}, \quad \delta_{ij} \sim \mathcal{N}(0, \sigma^2 I),$$

where $\mathcal{N}(0, \sigma^2 I)$ represents the multivariate normal distribution with a zero mean vector and a diagonal covariance matrix scaled by the variance σ^2, where $\sigma > 0$.

Solving an inverse problem with noisy observations $\tilde{y}_i$, introduces a perturbation to the target for the optimization problem specified by Definition 4.1, where we aim at minimizing

$$\|\tilde{y}_{n+1} - \hat{y}_{n+1}\| = \|y_{n+1} + \delta_{n+1} - \Phi_{h,f_\theta}(y_n + \delta_n)\|,$$

and $\tilde{y}_{n+1} - \hat{y}_{n+1} = 0$ is defined as the optimization target.

4.6.2 Motivation

The flow map $y(t_0 + h) = \varphi_{h,f}(y_0)$, of an ODE has the algebraic property, such that composing two steps with step size h_1 and h_2 equals one step with step size $h_1 + h_2$, or

$$\varphi_{f,h_1} \circ \varphi_{f,h_2}(y_0) = \varphi_{f,h_1+h_2}(y_0).$$

A numerical one-step method $\Phi_{h,f}$ is an approximation of the exact flow map up to order p. Hence, compositions of a one-step method could be utilized to generate multiple approximations to the same point in the flow, starting from different initial values. Assuming we know the points $\{\tilde{y}_0, \tilde{y}_1, \tilde{y}_2, \tilde{y}_3\}$, then $\hat{y}_2$ could be approximated in the three different ways

$$\begin{aligned}
\hat{y}_2^{(1)} &\approx \Phi_{h,f}(y_1), \\
\hat{y}_2^{(2)} &\approx \Phi_{h,f} \circ \Phi_{h,f}(y_0), \\
\hat{y}_2^{(-1)} &\approx \Phi_{-h,f}(y_3),
\end{aligned}$$

taking integrations steps with size h. For $\hat{y}_2^{(j)}$, the superscript (j) denotes the number of steps with length h in positive direction of the flow. If the numerical method used in the optimization problem is sensitive to a perturbation in a single point, taking the mean over all possible approximations using a fixed step size could reduce the sensitivity, as we will show below.

Let $\hat{f}_{n,n+1}$ be a simplified notation for how an inverse explicit numerical integrator evaluates the vector field, such that an integration step could be given by

$$\Phi_{h,f}(\tilde{y}_n, \tilde{y}_{n+1}) = \tilde{y}_n + h\hat{f}_{n,n+1}.$$

By repeatedly applying the inverse injection, compositions of integration steps could be approximated by summation over the vector field. For the midpoint method, we have that

$$\hat{f}_{n,n+1} = f\Big(\frac{\tilde{y}_n + \tilde{y}_{n+1}}{2}\Big),$$

such that taking two integration steps could be approximated by

$$\begin{aligned}
\hat{y}_2^{(2)} &= \Phi_{h,f} \circ \Phi_{h,f}(\tilde{y}_0) \\
&= \Phi_{h,f}(\tilde{y}_0) + hf\left(\frac{\Phi_{h,f}(\tilde{y}_0) + \hat{y}_2^{(2)}}{2}\right) \\
&\approx \Phi_{h,f}(\tilde{y}_0) + hf\left(\frac{\tilde{y}_1 + \tilde{y}_2}{2}\right) \\
&= \tilde{y}_0 + hf\left(\frac{\tilde{y}_0 + \Phi_{h,f}(\tilde{y}_0)}{2}\right) + h\hat{f}_{1,2} \\
&\approx \tilde{y}_0 + hf\left(\frac{\tilde{y}_0 + \tilde{y}_1}{2}\right) + h\hat{f}_{1,2} \\
&= \tilde{y}_0 + h\hat{f}_{0,1} + h\hat{f}_{1,2}.
\end{aligned}$$

This enables multiple vector field evaluations to be reused when computing the mean of the different flow approximations. In example, computing $\overline{y}_2$ could be done by

$$\overline{y}_2 = \frac{1}{3}\sum_i \hat{y}_2^{(i)} = \frac{1}{3}\bigg(\tilde{y}_0 + \tilde{y}_1 + \tilde{y}_3 + h(\hat{f}_{0,1} + 2\hat{f}_{1,2} - \hat{f}_{3,2})\bigg).$$

Assuming the ODE observation is given by S_N, then the mean approximation over the whole trajectory $\overline{y}_i$ for $i = 0, \dots, N$, could be computed simultaneously, reusing multiple vector field evaluations efficiently, as shown below.

4.6.3 The mean inverse integrator

Assume the vector field evaluation $\hat{f}_{i,j}$ is given by an inverse explicit integrator. The mean inverse integrator for $N = 3$ is given by

$$\begin{bmatrix} \overline{y}_0 \\ \overline{y}_1 \\ \overline{y}_2 \\ \overline{y}_3 \end{bmatrix} = \frac{1}{3}\left(\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{y}_0 \\ \tilde{y}_1 \\ \tilde{y}_2 \\ \tilde{y}_3 \end{bmatrix} + h \begin{bmatrix} -3 & -2 & -1 \\ 1 & -2 & -1 \\ 1 & 2 & -1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} \hat{f}_{0,1} \\ \hat{f}_{1,2} \\ \hat{f}_{2,3} \end{bmatrix} \right). \tag{4.7}$$

In comparison, a one-step scheme will be on the sparse form

$$\begin{bmatrix}\hat{y}_0\\ \hat{y}_1\\ \hat{y}_2\\ \hat{y}_3\end{bmatrix} = \begin{bmatrix}0&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}\tilde{y}_0\\ \tilde{y}_1\\ \tilde{y}_2\\ \tilde{y}_3\end{bmatrix} + h\begin{bmatrix}0&0&0\\1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}\hat{f}_{0,1}\\ \hat{f}_{1,2}\\ \hat{f}_{2,3}\end{bmatrix}. \tag{4.8}$$

In general, the mean inverse integrator is given by the following definition.

Definition 4.12 (Mean inverse integrator)
Let $\hat{f}_{ij}$ be the vector field evaluation of an inverse explicit one-step integrator. Assume that $\{y_i\}_{i=0}^N$ is a sample with $N+1$ sequential points with uniform step size h from an ODE with a vector field $f : \mathbb{R}^m \to \mathbb{R}^m$. Let

$$Y := [y_0, \dots, y_N]^T \in \mathbb{R}^{(N+1)\times m},$$
$$\hat{F} := [\hat{f}_{0,1}, \dots, \hat{f}_{N-1,N}]^T \in \mathbb{R}^{N\times m}.$$

The mean inverse integrator is given by

$$\overline{Y} = \frac{1}{N}\Big(UY + hV\hat{F}\Big), \tag{4.9}$$

such that $\overline{Y} := [\overline{y}_0, \dots, \overline{y}_N]^T \in \mathbb{R}^{(N+1)\times m}$, where $U \in \mathbb{R}^{(N+1)\times(N+1)}$ and $V \in \mathbb{R}^{(N+1)\times N}$ are defined by

$$[U]_{ij} := \begin{cases}0 & \textit{if} \quad i = j\\ 1 & \textit{else}\end{cases}$$
$$[V]_{ij} := \begin{cases}j-1-N & \textit{if} \quad j \geq i\\ j & \textit{else}\end{cases}$$

Considering Section 4.5, which illustrated different approaches to structuring the data for solving the optimization problem, the structure of the mean inverse integrator could be found in Figure 4.2.

In the setting of an inverse ODE problem, we assume that the vector field f is unknown. In that case, we replace $\hat{F}$ by

$$\hat{F}_\theta := \begin{bmatrix} f_\theta(s(y_0, y_1))\\ f_\theta(s(y_1, y_2))\\ \vdots\\ f_\theta(s(y_{N-1}, y_N))\end{bmatrix},$$

where $s(y_n, y_{n+1})$ is the vector field evaluation, such that $s(y_n, y_{n+1}) = \frac{1}{2}(y_n + y_{n+1})$ for the midpoint method, in example. Now denote the flow approximated by a neural network by

$$\overline{Y}_\theta := \frac{1}{N}\Big(UY + hV\hat{F}_\theta\Big). \tag{4.10}$$

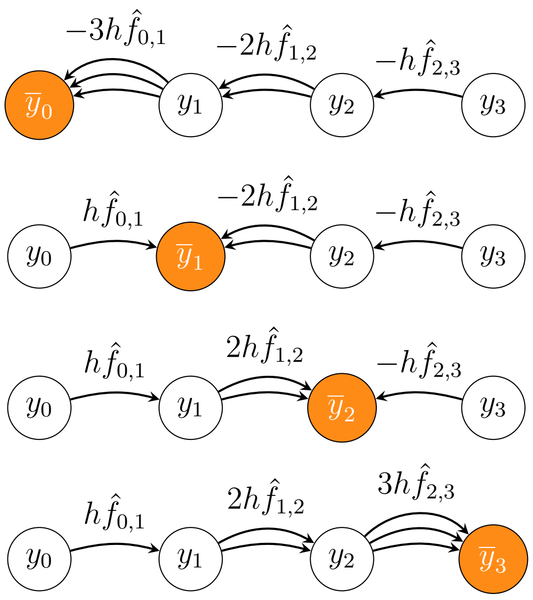


Figure 4.2: Illustration of the structure of the mean inverse integrator for $N = 3$.

If the observations are given by S_N^M where M initial values are observed in $N+1$ points in time, we let

$$Y^j := [y_0^j, \dots, y_n^j]^T \in \mathbb{R}^{(N+1)\times m}, \quad j = 1, \cdots, M,$$

and an inverse ODE problem using the mean inverse integrator is given by the optimization problem

$$\min_\theta \sum_{j=1}^{M} \|Y^j - \overline{Y}_\theta^j\|.$$

By Taylor expansion it is possible to compare how the mean inverse integrator propagates uncertainty to how a one-step method propagates uncertainty. In other words, if the observations S_N^M has a given variance σ^2, what will be the variance of the optimization target given by the mean inverse integrator

$$[Y]_i - [\overline{Y}_\theta]_i = \tilde{y}_i - \overline{y}_i,$$

compared to the optimization target obtained using a numerical one-step method directly

$$\tilde{y}_i - \hat{y}_i = \tilde{y}_i - \Phi_{h,f}(\tilde{y}_{i-1}).$$

Consider first a general result for any inverse explicit integration method where we assume that the step size in the flow sample h is small.

Theorem 4.13 (Variance in optimization target for small h)
Let $\{\tilde{y}_i\}_{i=0}^N$ be a noisy ODE observation with variance σ^2 following Definition 4.11. Let $\overline{y}_i = [\overline{Y}]_i$ be a step with the mean inverse integrator given by Definition 4.12 where $\hat{f}_{ij}$ is the vector field evaluation of an inverse explicit integration method $\hat{y}_{n+1} = \Phi_{h,f}(y_n)$.

Assuming the step size h is sufficiently small, the reduction of variance of the optimization target, when using the mean inverse integrator, compared to using a one-step method $\Phi_{h,f}$ could be approximated by

$$\frac{Var[\tilde{y}_i - \overline{y}_i]}{Var[\tilde{y}_i - \hat{y}_i]} \approx \frac{N+1}{2N}.$$

Proof. Assuming h is small, we can do the following truncation

$$\begin{aligned}\tilde{y}_i - \overline{y}_i &= \tilde{y}_i - \frac{1}{N}\sum_{\substack{j=0\\j\neq i}}^{N} \tilde{y}_j - \frac{h}{N}\sum_{j=1}^{N} v_{ij} f(s(\tilde{y}_j, \tilde{y}_{j+1})) \\ &\approx y_i + \delta_i - \frac{1}{N}\sum_{\substack{j=0\\j\neq i}}^{N}(y_j + \delta_j). \end{aligned} \tag{4.11}$$

We find an approximation to the variance by recalling that the perturbations are identically distributed $\delta_i \sim \mathcal{N}(0, \sigma^2 I)$ and independent, such that

$$\begin{aligned}\mathrm{Var}[\tilde{y}_i - \overline{y}_i] &\approx \mathrm{Var}[y_i + \delta_i] + \frac{1}{N^2}\mathrm{Var}\left[\sum_{\substack{j=0\\j\neq i}}^{N}(y_j + \delta_j)\right] \\ &= \mathrm{Var}[\delta_i] + \frac{1}{N}\mathrm{Var}[\delta_j] \\ &= \sigma^2 \frac{N+1}{N} I. \end{aligned} \tag{4.12}$$

Considering the one-step numerical integrator, we truncate by

$$\begin{aligned}\tilde{y}_i - \hat{y}_i &= y_i + \delta_i - y_{i-1} - \delta_{i-1} - hf(s(y_{i-1}, y_i) + s(\delta_{i-1}, \delta_i)) \\ &\approx y_i + \delta_i - y_{i-1} - \delta_{i-1}.\end{aligned}$$

Similarly, the variance could be approximated by

$$\begin{aligned}\mathrm{Var}[\tilde{y}_i - \hat{y}_i] &\approx \mathrm{Var}[y_i + \delta_i] + \mathrm{Var}[y_{i-1} - \delta_{i-1}] \\ &= \mathrm{Var}[\delta_i] + \mathrm{Var}[\delta_{i-1}] \\ &= 2\sigma^2 I, \end{aligned} \tag{4.13}$$

where $I \in \mathbb{R}^{m\times m}$ is the identity matrix. This leads to the following expression for the reduction of variance when using the mean inverse integrator in comparison with a one-step method:

$$\frac{\mathrm{Var}[\tilde{y}_i - \overline{y}_i]}{\mathrm{Var}[\tilde{y}_i - \hat{y}_i]} \approx \frac{N+1}{2N} I.$$

□

Secondly, we prove a more accurate approximation of the variance specifically for using the midpoint method in the mean inverse integrator.

Theorem 4.14 (Variance in optimization target for the midpoint method for large h) *The variance in the optimization target of the midpoint method in mean inverse integrator $\overline{y}_i$, and the variance in the optimization target using the midpoint method directly $\hat{y}_i$, could be estimated by*

$$Var[\tilde{y}_i - \overline{y}_i] \approx \frac{\sigma^2}{N}\Bigg[(N+1)I - \frac{h}{2}(P_{i,i} + P_{i,i}^T) + \frac{h}{2N}\sum_{\substack{j=0\\ j\neq i}}^{N}(P_{i,j} + P_{i,j}^T) + \frac{h^2}{4N}\Bigg(\sum_{j=1}^{N} P_{i,j}P_{i,j}^T\Bigg)\Bigg],$$

and

$$Var[\tilde{y}_i - \hat{y}_i] \approx \sigma^2\Bigg[2I + \frac{h^2}{2}f'\Big(\frac{y_{i-1}+y_i}{2}\Big)f'\Big(\frac{y_{i-1}+y_i}{2}\Big)^T\Bigg],$$

for $i = 0, \dots, N$ where

$$P_{i,j} := \tilde{v}_{i,j}f'\Big(\frac{y_{j-1}+y_j}{2}\Big) + \tilde{v}_{i,j+1}f'\Big(\frac{y_j+y_{j+1}}{2}\Big),$$

and $\tilde{v}_{i,j} = [\tilde{V}]_{i,j}$ where $\tilde{V} \in \mathbb{R}^{(N+1)\times(N+2)}$ is obtained from padding each row of V with zeros in the beginning and the end, and allowing the columns of $\tilde{V}$ to be zero-indexed.

Proof. For the midpoint method, we have that $s(y_i, y_{i+1}) := \frac{y_i+y_{i+1}}{2}$. Consider the Taylor expansion of the mean inverse integrator in the direction of the perturbation, such that

$$\begin{aligned}
\tilde{y}_i - \overline{y}_i &= y_i + \delta_i - \frac{1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}(y_j+\delta_j) - \frac{h}{N}\sum_{j=1}^{N} v_{ij} f\Big(s(y_{j-1},y_j) + s(\delta_{j-1},\delta_j)\Big)\\
&\approx y_i + \delta_i - \frac{1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}(y_j+\delta_j) - \frac{h}{N}\sum_{j=1}^{N} v_{ij}\Big[f(s(y_{j-1},y_j)) + f'(s(y_{j-1},y_j))s(\delta_{j-1},\delta_j)\Big]\\
&= y_i + \delta_i - \frac{1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}(y_j+\delta_j) - \frac{h}{N}\sum_{j=1}^{N} v_{ij}\Big[f\Big(\frac{y_{j-1}+y_j}{2}\Big) + f'\Big(\frac{y_{j-1}+y_j}{2}\Big)\frac{\delta_{j-1}+\delta_j}{2}\Big].
\end{aligned}$$

We collect the terms where δ_i and δ_j appears and introduce the matrices $P_{i,j} \in \mathbb{R}^{s\times s}$ depending on the Jacobian f', such that

$$\begin{aligned}
\frac{-1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}\Big(I + \frac{h}{2}P_{i,j}\Big)\delta_j &:= \frac{-1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}\Bigg[I + \frac{h}{2}\Bigg(\underbrace{\tilde{v}_{i,j}f'\Big(\frac{y_{j-1}+y_j}{2}\Big) + \tilde{v}_{i,j+1}f'\Big(\frac{y_j+y_{j+1}}{2}\Big)}_{=:P_{i,j}}\Bigg)\Bigg]\delta_j,\\
\Big(I - \frac{h}{2N}P_{i,i}\Big)\delta_i &= \Bigg[I - \frac{h}{2N}\Bigg(\underbrace{\tilde{v}_{i,i}f'\Big(\frac{y_{i-1}+y_i}{2}\Big) + \tilde{v}_{i,i+1}f'\Big(\frac{y_i+y_{i+1}}{2}\Big)}_{=P_{i,i}}\Bigg)\Bigg]\delta_i,
\end{aligned}$$

where $i, j = 0, \dots, N$. The coefficient matrix $\tilde{V} \in \mathbb{R}^{(N+1)\times(N+2)}$ is obtained from padding each row of V with zeros in the beginning and end, such that $\tilde{v}_{i,j} = 0$ if

$j \in \{0, N\}$ and $\tilde{v}_{i,j} = v_{i,j}$ else. Furthermore, the columns of $\tilde{V}$ are zero indexed, in contrast to the notation for matrices in the rest of the thesis.

Note that for multivariate distributions X, we have that $\mathrm{Var}[X] := \mathrm{Cov}[X, X]$ where the covariance $\mathrm{Cov}[\cdot\,,\,\cdot]$ is a bilinear form. As δ_i is independent of δ_j we get

$$\begin{aligned}
\mathrm{Var}[\tilde{y}_i - \overline{y}_i] \approx& \mathrm{Var}\left[\left(I - \frac{h}{2N}P_{i,i}\right)\delta_i\right] + \mathrm{Var}\left[\frac{-1}{N}\sum_{\substack{j=0\\ j\neq i}}^{N}\left(I + \frac{h}{2}P_{i,j}\right)\delta_j\right] \\
=& \sigma^2\left(I - \frac{h}{2N}P_{i,i}\right)\left(I - \frac{h}{2N}P_{i,i}\right)^T + \frac{\sigma^2}{N^2}\sum_{\substack{j=0\\ j\neq i}}^{N}\left(I + \frac{h}{2}P_{i,j}\right)\left(I + \frac{h}{2}P_{i,j}\right)^T \\
=& \frac{\sigma^2}{N}\Bigg[(N+1)I - \frac{h}{2}(P_{i,i} + P_{i,i}^T) + \frac{h}{2N}\sum_{\substack{j=0\\ j\neq i}}^{N}(P_{i,j} + P_{i,j}^T) \\
&+ \frac{h^2}{4N}\Bigg(P_{i,i}P_{i,i}^T + \sum_{\substack{j=0\\ j\neq i}}^{N} P_{i,j}P_{i,j}^T\Bigg)\Bigg] \\
=& \frac{\sigma^2}{N}\Bigg[(N+1)I - \frac{h}{2}(P_{i,i} + P_{i,i}^T) + \frac{h}{2N}\sum_{\substack{j=0\\ j\neq i}}^{N}(P_{i,j} + P_{i,j}^T) + \frac{h^2}{4N}\Bigg(\sum_{j=1}^{N} P_{i,j}P_{i,j}^T\Bigg)\Bigg].
\end{aligned} \tag{4.14}$$

For the midpoint method we find the optimization target by

$$\begin{aligned}
\tilde{y}_i - \hat{y}_i &\approx \delta_i - \delta_{i-1} - hf'\Big(\frac{y_{i-1} + y_i}{2}\Big)\frac{\delta_{i-1} + \delta_i}{2} \\
&= \left(I - \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\delta_i - \left(I + \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\delta_{i-1},
\end{aligned}$$

so that the variance could be found by

$$\begin{aligned}
\mathrm{Var}[\tilde{y}_i - \hat{y}_i] \approx& \mathrm{Var}\left[\left(I - \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\delta_i\right] + \mathrm{Var}\left[\left(I + \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\delta_{i-1}\right] \\
=& \sigma^2\left(I - \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\left(I - \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)^T \\
&+ \left(I + \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)\left(I + \frac{h}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)\right)^T \\
=& \sigma^2\left[2I + \frac{h^2}{2}f'\Big(\frac{y_{i-1} + y_i}{2}\Big)f'\Big(\frac{y_{i-1} + y_i}{2}\Big)^T\right].
\end{aligned} \tag{4.15}$$

□

Remark

Bounds on the variance estimates could be found by assuming, for some matrix norm, that the Jacobian $\|f'(y_i)\| \leq C$ *is bounded* $\forall y_i \in S \subset \mathbb{R}^m$ *for a given problem. However,* *if this is done for an inverse problem, the vector field f is by definition unknown, and it would thus be uncertain if such an assumption is reasonable.*

Consider the variance estimate for using the midpoint method as a one-step integrator, and note that the first order terms of h are cancelled in Equation (4.15). This happens as the midpoint integrator evaluates the vector field in the midpoint. This is note true for all symmetric integrators. Consider for instance the implicit trapezoidal rule, which is symmetric. However, in this case the first order term in h, would not cancel in the corresponding variance estimate.

Instead of taking the mean of the different numerical integration trajectories in the mean inverse integrator by Definition 4.12, one could in general compute a weighted sum of trajectories $\overline{y}_k = \sum_i w_i \hat{y}_k^{(i)}$ where $\sum_i w_i = 1$. Here, the weights w_i could be chosen as to minimize the variance and the mean error of the optimization target.

Chapter 5

Numerical experiments

5.1 Experimental setup

5.1.1 Implementation

The code implemented to run the numerical experiments described below, contains the following main modules:

- A general class for representing a Hamiltonian system which is instantiated by a LaTeX-coded Hamiltonian and utilizes SymPy for computing gradients of the Hamiltonian to obtain the vector field $f(y)$.
- A function to generate the flow sample S_N^M for a given Hamiltonian system, with or without noise, used for training the neural networks.
- A class for representing general Runge–Kutta and MIRK methods, in addition to functions to do implicit integration with the fixed point iterations described in Section 4.4. All computations are implemented using PyTorch arrays such that numerical operations are recorded in a computational graph, enabling automatic differentiation.
- Routines for training neural networks (implemented in PyTorch) using the flow sample S_N^M and a numerical integrator $\Phi_{h,f}$ which could be used as a mean inverse integrator or as a one-step method.
- Specific routines for running and plotting the results from the experiments described in this section.

The modules described above depend are implemented in Python and builds on several libraries:

- PyTorch by Paszke et al. (2019) is used to implement the neural networks approximating the vector field f_θ. The library allows for efficient computation of gradients implementing an algorithm for automatic differentiation. It also contains implementations of several optimization routines, such as L-BFGS and Adam.

- NumPy by Harris et al. (2020) is used together with PyTorch to handle vectors, matrices and tensors.
- SciPy by Virtanen et al. (2020) has implementations of several numerical integrators and is used for generating the training data, where integration over the Hamiltonian systems is necessary. Specifically, the Dormand-Prince, Runge–Kutta method of order $p = 8$ Dormand and Prince (1980) with step size control, abbreviated by *DOP853* is used.
- SymPy by Meurer et al. (2017) has been useful to do symbolic computations. Avoiding hard-coding every Hamiltonian for testing, SymPy allows for parsing a LaTeX-coded Hamiltonian into a Python-function operating on NumPy arrays. Furthermore, it allows for symbolic differentiation and matrix operations useful for testing and mathematical exploration.
- PyBs by Sundklakk (2015) is a Python library for handling B-series and thus for checking order conditions and symplecticity of Runge–Kutta integration methods. This was actively used for exploring different Runge–Kutta methods.
- The discrete gradient algorithm for neural networks and the PyTorch implementation by Matsubara et al. (2019) including modifications as presented by Eidnes (2022) were used for the numerical experiments.

5.1.2 Numerical integrators used in training

Table 5.1 provides an overview of the different numerical integration methods presented in the thesis and used in experiments.

Integration method	Name in plots	Order	Symm.	Sympl.	Inv. expl.	Type
Symplectic Euler	Symplectic Euler	1	no	yes	yes	PRK
Implicit midpoint	Midpoint	2	yes	yes	yes	MIRK
MIRK3	MIRK3	3	no	no	yes	MIRK
MIRK4	MIRK4	4	no	no	yes	MIRK
MIRK4 from midpoint	MIRK4 mid	4	yes	no	yes	MIRK
MIRK6	MIRK6	6	no	no	yes	MIRK
MIRK3 Symmetrized	MIRK3 sym	4	yes	no	yes	MIRK
MIRK5 Symmetrized	MIRK5 sym	6	yes	no	yes	MIRK
Gauss col. $s = 2$	GC4	4	yes	yes	no	RK
Gauss col. $s = 3$	GC6	6	yes	yes	no	RK
Discrete gradient 2	DGM2	2	yes	no	yes	DG
Discrete gradient 3	DGM3	3	no	no	yes	DG

Table 5.1: Numerical integration methods used in the experiments. *Symm.* is short for symmetric, *sympl.* for symplectic, *col.* for collocation. *PRK* is short for partitioned Runge–Kutta and *DG* for discrete gradients.

The midpoint method could be found in (Hairer et al., 2006, p. 30) whereas symplectic Euler is introduced in (Hairer et al., 2006, p. 189). The MIRK methods are first introduced by Burrage et al. (1994). MIRK3 and MIRK4 is presented in Section 2.5.2. MIRK5 (used in the symmetric method) is found by taking $c_2 = 0$ and $c_3 = \frac{4}{3}$, whereas for MIRK6 we choose $c_3 = \frac{1}{4}$ and $c_4 = \frac{3}{4}$ considering how the methods are presented by Burrage et al. (1994). The symmetrized methods are presented in Example 4.10. The Gauss collocation methods was introduced in Section 2.5.1 and taken from (Hairer et al., 2006, p. 34). Finally, the discrete gradients methods are found by Eidnes (2022).

5.1.3 Hamiltonian systems

These definitions follow the report for the specialization project, Noren (2022). The following four Hamiltonian systems will be included in the numerical experiments:

Definition 5.1 (Pendulum)
Let the pendulum problem be defined setting the mass $m = 1$, the pendulum length $l = 1$ and the acceleration due to gravitation to $g = 1$. Let $q = \theta$ denote the angle and p the angular momentum of the pendulum. The pendulum problem then has a separable Hamiltonian and is on the form

$$H(q_1, p_1) = \frac{1}{2}p_1^2 + (1 - \cos(q_1)).$$

Definition 5.2 (Double pendulum)
The double pendulum system has a Hamiltonian that is not separable, where

$$y = [q_1, q_2, p_1, p_2]^T.$$

Let q_i and p_i denote the angle and angular momentum of pendulum $i = 1, 2$. The double pendulum is a chaotic system, meaning a small perturbation to the initial value y_0 yields potentially significant changes in the resulting flow. An introduction to chaotic dynamics could be found in (Goldstein et al., 2001, Chapter 11). The Hamiltonian is given by

$$H(q_1, q_2, p_1, p_2) = \frac{\frac{1}{2}p_1^2 + p_2^2 - p_1 p_2 \cos(q_1 - q_2)}{1 + \sin^2(q_1 - q_2)} - 2\cos(q_1) - \cos(q_2).$$

Definition 5.3 (Hénon-Heiles)
The Hénon-Heiles model was introduced for describing stellar motion inside the gravitational potential of a galaxy, as described in (Hairer et al., 2006, Chapter I.3). This Hamiltonian is separable. However, it is a canonical example of a chaotic system and its properties are discussed more in detail in (Goldstein et al., 2001, Chapter 11). The Hamiltonian is given by

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

5.1.4 Experimental parameters

The parameters needed in to specify the numerical experiments is given below:

- The analytical Hamiltonian $H : \mathbb{R}^m \to \mathbb{R}$.
- The integration parameters:
  - $r > 0$, the radii in the ball $S_0 = \{y_0 \text{ s.t. } \|y_0\|_2 \leq r\}$ where initial values for generating the flow sample S_N^M are sampled.
  - $N \in \mathbb{Z}_{>0}$, the number of partitions of the time interval in the flow sample S_N^M for training, such that $N + 1$ is the number of points in time.
  - $M \in \mathbb{Z}_{>0}$, the number of initial values in the flow sample S_N^M.
  - $h > 0$, the step length in the training data.
  - TOL, the tolerance for terminating the fixed-point iterations of the implicit integrators.
  - $n_{\text{max iter fp}}$, the maximum number of iterations allowed for the fixed-point iterations.
- The neural network and optimization parameters:
  - $L \in \mathbb{Z}_{>0}$, the number of layers.

- $\{n_i\}_{i=1}^{L-1}$, the set of dimensions for the linear layers.
- $\{\sigma_i\}_{i=0}^{L}$, the set of differentiable non-linear activation functions.
- n_{history}, the history size of L-BFGS.
- $n_{\text{max iter}}$ the maximum number of iterations per time step.
- n_{epochs}, the number of training epochs.

The neural networks and their corresponding optimization procedure used in the the numerical experiments below, are defined by the following parameters.

$$\begin{aligned}
L &= 4 \\
\sigma_i &= \tanh \\
\{n_i\}_{i=0}^{4} &= \{m, n_{\text{hidden}}, n_{\text{hidden}}, 1\} \\
n_{\text{history}} &= 10 \\
n_{\text{max iter}} &= 15
\end{aligned}$$

where n_{hidden} is a positive integer specifying the dimension of the hidden layers and m is the dimension of the ODE such that $y_i \in \mathbb{R}^m$. The hidden dimension is set to $n_{\text{hidden}} = 100$ for problems where $y_i \in \mathbb{R}^2$ and to $n_{\text{hidden}} = 200$ for problems where $y_i \in \mathbb{R}^4$. Furthermore, the implicit integrators are always used with

$$\begin{aligned}
\text{TOL} &= 10^{-14} \\
n_{\text{max iter fp}} &= 30.
\end{aligned}$$

5.2 Variance in the optimization target

Here, assuming that the vector field f is known, we aim at testing if the two different approximations to the variance, or the covariance matrices of the optimization target, provided by Theorem 4.13 and Theorem 4.14, are accurate.

Let $y_i = \varphi_{ih,f}(y_0) + \delta_i$ be a random variable since we assume that $\delta_i \sim \mathcal{N}(0, \sigma^2 I)$ for $i = 0, \cdots, N$. In order to check if the variance estimates given by Theorem 4.13 and Theorem 4.14 are accurate, we will derive an empirical estimate of the variance by sampling. We draw $(N+1)\cdot K$ realizations ϵ_{ij} from the normal distribution $\mathcal{N}(0, \sigma^2 I)$ such that

$$\begin{aligned}
\tilde{y}_i^j &= y_i + \epsilon_{ij} \\
\text{and} \quad \tilde{Y}^j &= [\tilde{y}_0^j, \ldots, \tilde{y}_n^j]^T \in \mathbb{R}^{(N+1)\times m}, \quad j = 1, \cdots, K.
\end{aligned}$$

Hence, we can estimate the distribution of the points found when integrating over $\tilde{y}_i^j$, first by the mean inverse integrator and then using the one-step approach. Let $\Phi_{h,f}^{\text{OS}}(\tilde{y}_{i-1}^j)$ denote the j-th approximation of $y(t_i)$ using the one-step approach, and $\Phi_{h,f}^{\text{MII}}(\tilde{Y}^j)_i$ denote the j-th approximation of $y(t_i)$ using the mean inverse integrator.

Assuming we have K samples $x^j \in \mathbb{R}^m$, for $j = 1, \dots, K$ or as a matrix we have $x \in \mathbb{R}^{m \times K}$. The mean vector $\overline{x} \in \mathbb{R}^m$ and the covariance matrix $Q \in \mathbb{R}^{m \times m}$ of this sample could be estimated by

$$\overline{x} := \frac{1}{K} \sum_{j=0}^{N} x^j, \tag{5.1}$$

$$Q(x) := \frac{1}{K-1} \sum_{j=0}^{K} \left(x^j - \overline{x}\right)\left(x^j - \overline{x}\right)^T. \tag{5.2}$$

In the same fashion, we can estimate the covariance matrix of the K approximations to $y(t_i)$ using both integration methods. Let

$$Q_i^{\mathrm{OS}} := Q(\Phi_{h,f}^{\mathrm{OS}}(y_{i-1}))$$
$$Q_i^{\mathrm{MII}} := Q(\Phi_{h,f}^{\mathrm{MII}}(\tilde{Y})_i),$$

be the sample covariance matrices calculated using K samples where $\Phi_{h,f}^{\mathrm{OS}}(y_{i-1})$ and $\Phi_{h,f}^{\mathrm{MII}}(\tilde{Y})_i$ are matrices in $\mathbb{R}^{m \times K}$. Let

$$\hat{Q}_i^{\mathrm{OS}} \quad \text{and} \quad \hat{Q}_i^{\mathrm{MII}},$$

be the approximations for large h by Theorem 4.14 and let

$$\overline{Q}_i^{\mathrm{OS}} \quad \text{and} \quad \overline{Q}_i^{\mathrm{MII}},$$

denote the approximations where we assume that h is small, by Equation (4.13) and (4.12). To obtain a more practical measure of variance, instead of reporting the covariance matrices, we will compute the spectral radius of the three different covariance matrix approximations. The spectral radius is the largest absolute value of the eigenvalues:

Definition 5.4 (Spectral radius)
Let $\lambda_1, \dots, \lambda_n$ be the eigenvalues of a matrix $A \in \mathbb{R}^{n \times n}$. The spectral radius is defined as

$$\rho(A) := \max\{|\lambda_1|, \dots, |\lambda_n|\}.$$

Let $\rho_i^{\mathrm{INT}} := \rho(Q_i^{\mathrm{INT}})$ where $\mathrm{INT} \in \{\mathrm{OS}, \mathrm{MII}\}$. For testing if the variance estimates are accurate, we will assume that the vector field f is known and that one trajectory $y_i \approx \varphi_{ih,f}(y_0)$ is given. Furthermore, normally distributed perturbations are sampled, to obtain $\tilde{y}_i^j$ for $i = 0, \dots, N$ and $j = 1, \dots, K$. Then, integration over the K trajectories allows for computing the sample variances and the corresponding spectral radii, and secondly by using the known vector field and its Jacobian f', the estimates for small and large h could be computed.

This is first done for fixed step sizes, plotting the of the spectral radii over time $t_i = ih$ for $i = 0, \dots, N$. Secondly, by letting $h_k = \frac{1}{i}$ for $i = 1, \dots, K$, and computed the mean of spectral radius over one flow by

$$\frac{1}{N+1} \sum_{i=0}^{N} \rho_i^{\text{INT}}.$$

Finally, since the spectral radius of the covariance matrix represents the variance, computing the square root $\sqrt{\rho}$ of this quantity would represent the standard deviation and would be on the same order of magnitude as the mean, allowing for easier comparison and analysis.

Algorithm 1 Variance estimates over one trajectory fixed h

```
Require: Vector field $f$, a trajectory $\{y_i\}_{i=0}^N$ with step size $h$, variance $\sigma^2$.
  Draw $\epsilon_{ij}$ from $\mathcal{N}(0, \sigma^2 I)$ to get $\tilde{y}_i^j = y_i + \epsilon_{ij}$
  Compute sample variance $\rho_i^{\text{OS}}, \rho_i^{\text{MII}}$ from integration over $f$
  Compute the small $h$ estimate $\overline{\rho}_i^{\text{OS}}, \overline{\rho}_i^{\text{MII}}$ by Equations (4.12) and (4.13).
  Compute the large $h$ estimate $\hat{\rho}_i^{\text{OS}}, \hat{\rho}_i^{\text{MII}}$ by Equations (4.14) and (4.15).
```

Results running Algorithm 1 on the simple and double pendulum, including the Henon–Héiles ODE for $h = 0.1, 0.5$, with $\sigma = 0.05$, for a trajectory with $N = 6$ and sampling $K = 10^5$ samples, could be found in Figure 5.1.

Algorithm 2 Variance estimates with decreasing h

```
Require: Vector field $f$, trajectories $S_N^j = \{y_i\}_{i=0}^N$ for decreasing step sizes $h_j$, variance
  $\sigma^2$.
  for $k = 1, \dots, K$ do
      $\{y_i\}_{i=0}^N \leftarrow S_N^k$
      $h \leftarrow h_k$
      Draw $\epsilon_{ij}$ from $\mathcal{N}(0, \sigma^2 I)$ to get $\tilde{y}_i^j = y_i + \epsilon_{ij}$
      Compute the mean sample variance $\rho^{\text{OS}}, \rho^{\text{MII}}$ from integration over $f$
      Compute the mean small $h$ estimate $\overline{\rho}^{\text{OS}}, \overline{\rho}^{\text{MII}}$ by Equations (4.12) and (4.13).
      Compute the mean large $h$ estimate $\hat{\rho}^{\text{OS}}, \hat{\rho}^{\text{MII}}$ by Equations (4.14) and (4.15).
  end for
```

Letting again $N = 6$, $\sigma = 0.05$ and $K = 10^5$, the experiment described by Algorithm 2 is run for $h_k = \left\{1,\ 0.9,\ 0.8,\ \dots, 0.1\right\}$ for simple and double pendulum. For Henon–Héiles, we let $h_k = \left\{0.7,\ 0.6,\ \dots, 0.1\right\}$ due to the large error when too large step sizes are used for this chaotic dynamical system. The results are found in Figure 5.2.

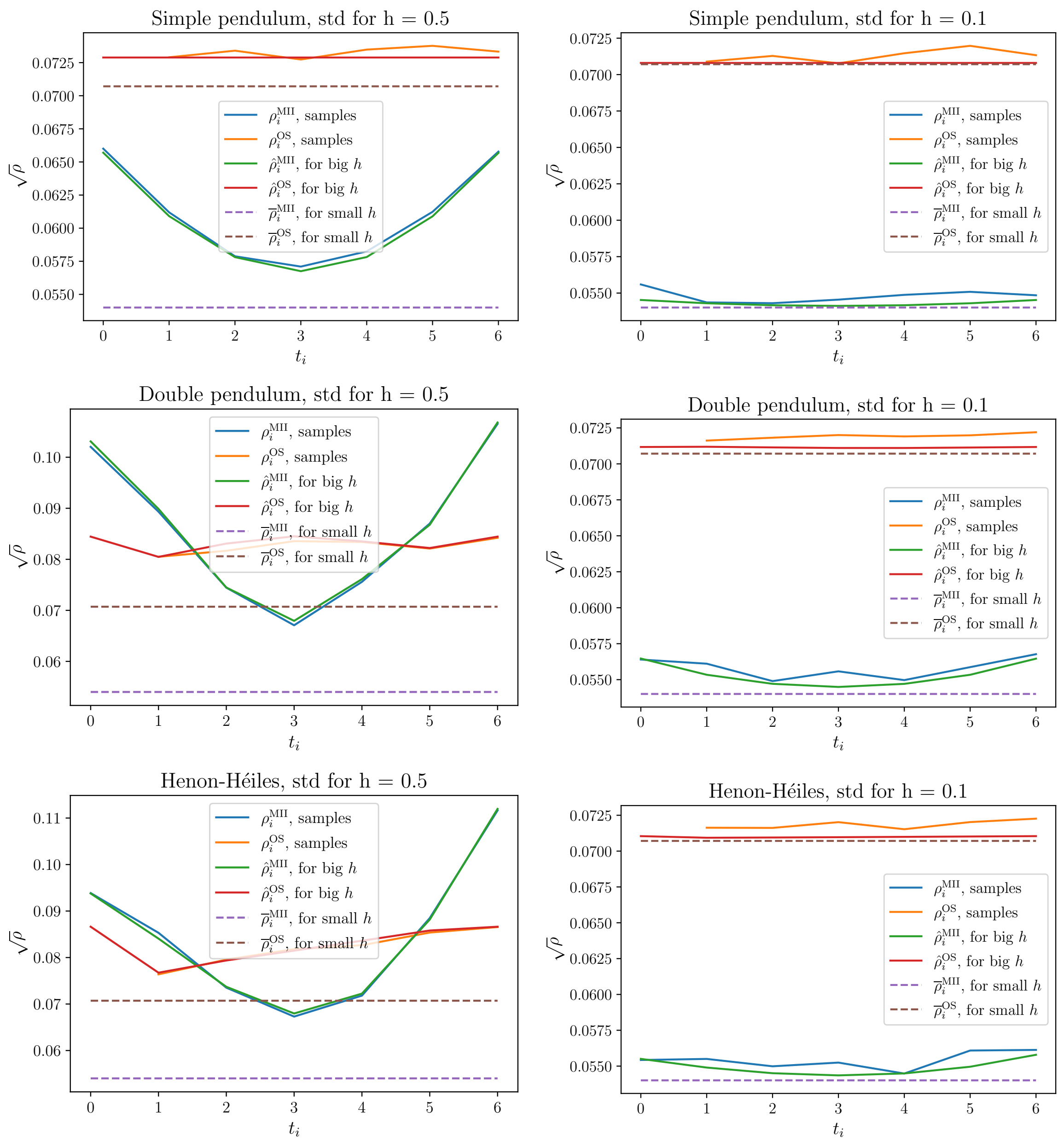


Figure 5.1: Results for the experiment following Algorithm 1. Three different estimates of the standard deviation $\sqrt{\rho}$ over the flow when integrating over noisy initial values, for $t_0, \ldots, t_6$ on the x-axis. The dotted lines represent the estimate assuming h is small, the red and green lines are the more accurate estimate where h is not negligible, whereas the blue and orange are estimated by sampling. Simple pendulum (top row), double pendulum (mid row) and Henon–Héiles (bottom row) tested for $h = 0.1$ (left column) and $h = 0.5$ (right column).

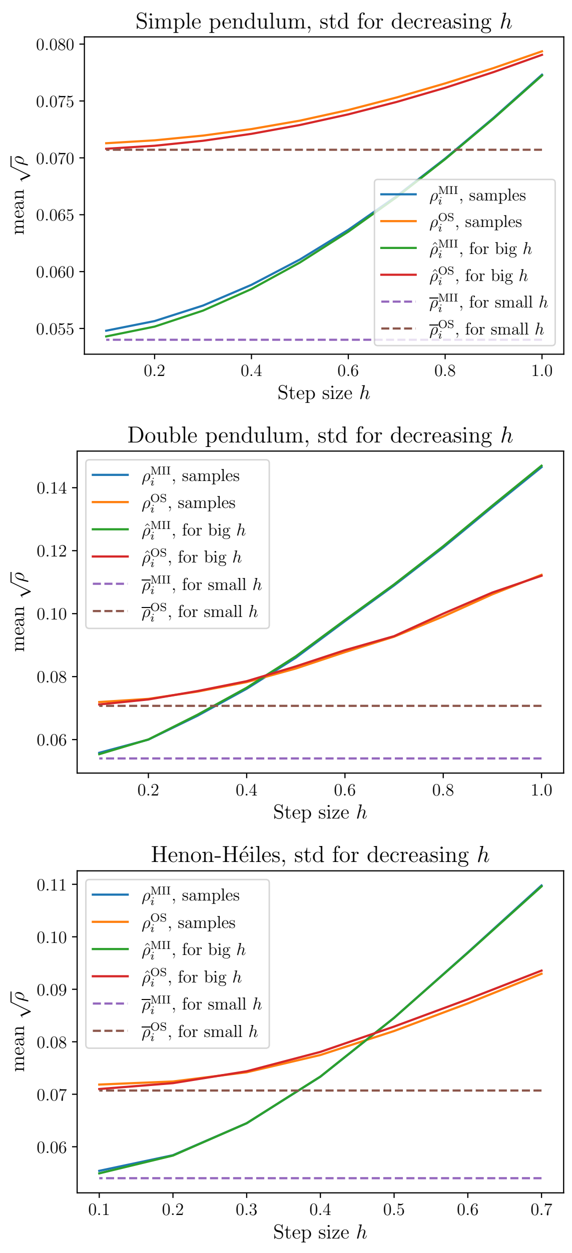


Figure 5.2: Results for the experiment following Algorithm 2. Three different estimates of the standard deviation $\sqrt{\rho}$ of the integration error for decreasing step sizes h. Simple pendulum (top), double pendulum (mid) and Henon–Héiles (bottom).

5.3 Variance after training

This experiment aims at testing how noise in the data is propagated by an integration method to the learned vector field f_θ. In other words: it aims at testing if the mean inverse integrator could enable learning a vector field more accurately, when there is noise in the data, compared to using a one-step method.

Let S be a tensor with points in the flow such that $S \in \mathbb{R}^{(N+1)\times M\times m}$ and

$$[S]_{ij} := \varphi_{ih,f}(y_0^j)$$

where $i = 0, \dots, N$ and $j = 1, \dots, M$. For approximating the normal distribution, let $\Delta^l \in \mathbb{R}^{(N+1)\times M\times m}$ be a tensor of independent samples ϵ_{ijl} from the normal distribution $\mathcal{N}(0, \sigma^2 I)$ such that we get multiple perturbed flow samples $\tilde{S}^l$ by

$$\begin{aligned} [\tilde{S}^l]_{ij} &:= \varphi_{ih,f}(y_0^j) + \epsilon_{ijl} \\ \iff \tilde{S}^l &= S + \Delta^l, \end{aligned}$$

for $l = 1, \dots, L$. In order to investigate how different training procedures propagate noise, the mean inverse integrator and the one-step method will be used to train multiple neural networks, one each for each integration method, for each perturbed flow sample $\tilde{S}^l$. This gives a set of neural networks $\{f_{\theta,l}^{\text{MII}}, f_{\theta,l}^{\text{OS}}\}_{l=1}^{L}$ and the noise propagation could be studied by integrating over all neural networks from the same initial value, and study the distribution of the end point. Let the approximated flow end points be given by

$$\begin{aligned} y_{\tilde{N},l}^{\text{MII}} &:= \Phi_{h,f} \circ \dots \circ \Phi_{h,f}(y_0), \qquad f = f_{\theta,l}^{\text{MII}}, \\ y_{\tilde{N},l}^{\text{OS}} &:= \Phi_{h,f} \circ \dots \circ \Phi_{h,f}(y_0), \qquad f = f_{\theta,l}^{\text{OS}}, \end{aligned}$$

where $\tilde{N}$ is the number of integration steps in the testing. However, as we are only interested in how perturbations of the flow sample S is propagated during neural network training, the interesting quantity is the error

$$\begin{aligned} e_{\tilde{N}}^{\text{MII}} &:= [y_{\tilde{N},1}^{\text{MII}} - y_{\tilde{N}}, \dots, y_{\tilde{N},L}^{\text{MII}} - y_{\tilde{N}}]^T, \\ e_{\tilde{N}}^{\text{OS}} &:= [y_{\tilde{N},1}^{\text{OS}} - y_{\tilde{N}}, \dots, y_{\tilde{N},L}^{\text{OS}} - y_{\tilde{N}}]^T, \end{aligned}$$

where $y_{\tilde{N}} \approx \varphi_{\tilde{N}h,f}(y_0)$ is found by integrating over the true vector field f and $e_{\tilde{N}}^{\text{MII}}, e_{\tilde{N}}^{\text{OS}} \in \mathbb{R}^{L\times m}$. The magnitude of the covariance of the error is estimated by the sample covariance following Equation (5.1) and computing the spectral radius by Definition 5.4, yielding

$$\begin{aligned} \rho^{\text{MII}} &:= \rho(Q(e_{\tilde{N}}^{\text{MII}})), \\ \rho^{\text{OS}} &:= \rho(Q(e_{\tilde{N}}^{\text{OS}})). \end{aligned}$$

Finally, it is not only the variance in the flow approximation over the learned vector fields that is relevant, but also the mean, which is found by

$$\overline{e}^{\text{MII}} := \left\| \frac{1}{L} \sum_{l=1}^{L} e_{\tilde{N},l}^{\text{MII}} \right\|,$$
$$\overline{e}^{\text{OS}} := \left\| \frac{1}{L} \sum_{l=1}^{L} e_{\tilde{N},l}^{\text{OS}} \right\|.$$

In order to study how the noise is propagated, multiple experiments will be conducted with decreasing step size h in the flow sample S. Furthermore, to make the experiments more robust, we compute the mean of ρ^{MII}, ρ^{OS}, $\overline{e}^{\text{MII}}$ and $\overline{e}^{\text{OS}}$ for $\tilde{M}$ different initial values.

Algorithm 3 Variance after training for decreasing h

```
Require: Vector field $f$, trajectories $S^k = \{y_i^j\}_{i=0,j=1}^{N,M}$ for decreasing step sizes $h_k$,
  variance $\sigma^2$.
  for $k = 1, \dots, K$ do
      $h \leftarrow h_k$
      $S \leftarrow S^k$
      for $l = 1, \dots L$ do
          Draw $\epsilon_{ijl}$ from $\mathcal{N}(0, \sigma^2 I)$ to get $\tilde{S}^l = S + \Delta^l$
          Train neural network $\min_\theta \mathcal{L}(\Phi^{\text{MII}}, \tilde{S}^l, f_{\theta,l}^{\text{MII}})$
          Train neural network $\min_\theta \mathcal{L}(\Phi^{\text{OS}}, \tilde{S}^l, f_{\theta,l}^{\text{OS}})$
      end for
      Compute mean $e_N^{\text{MII}}$ and $e_N^{\text{OS}}$ integrating over $f_{\theta,l}^{\text{MII}}$ and $f_{\theta,l}^{\text{OS}}$ over $M$ initial values.
      Compute mean $\rho_k^{\text{MII}}$ and $\rho_k^{\text{OS}}$ over $\tilde{M}$ initial values.
      Compute mean $\overline{e}_k^{\text{MII}}$ and $\overline{e}_k^{\text{OS}}$ over $\tilde{M}$ initial values.
  end for
```

The experiment in Algorithm 3 is run for $L = 10$ noisy realizations of the flow sample with S given by $N = 6$ partitions, $M = 500$ different initial values that are bounded $\|y_0^j\| \leq r$ with $r = 0.7$. The neural networks $f_{\theta,l}^{\text{OS}}$ are trained with $n_{\text{epochs}} = 6$, whilst $f_{\theta,l}^{\text{MII}}$ are trained 3 epochs with the one-step method and then 3 epochs using the mean inverse integrator. Similar as the previous experiment, we have $h_k = \Big\{1,\ 0.9,\ 0.8,\ \dots, 0.1\Big\}$ for the simple pendulum problem. However, for double pendulum we consider $h \in [0.7, 0.1]$ and for Henon–Héiles we let $h \in [0.5, 0.1]$, due too the large error obtained when choosing to large step sizes h for chaotic systems. For all problems, let $\sigma = 0.05$. Finally, we compute the distribution after integrating $\tilde{M} = 25$ initial values $\tilde{N} = 5$ steps. The results for the simple and double pendulum, in addition to the Henon–Héiles problem can be found in Figure 5.3.

The plots in the figures 5.4, 5.5 and 5.6, visualizes the distribution of points when integrating over the learned vector fields. The networks are trained in the same manner as above, with step size $h = 0.1$ where $\sigma = 0.05$. After training the neural networks, $\tilde{N} = 60$ integration steps are done, to generate the plots.

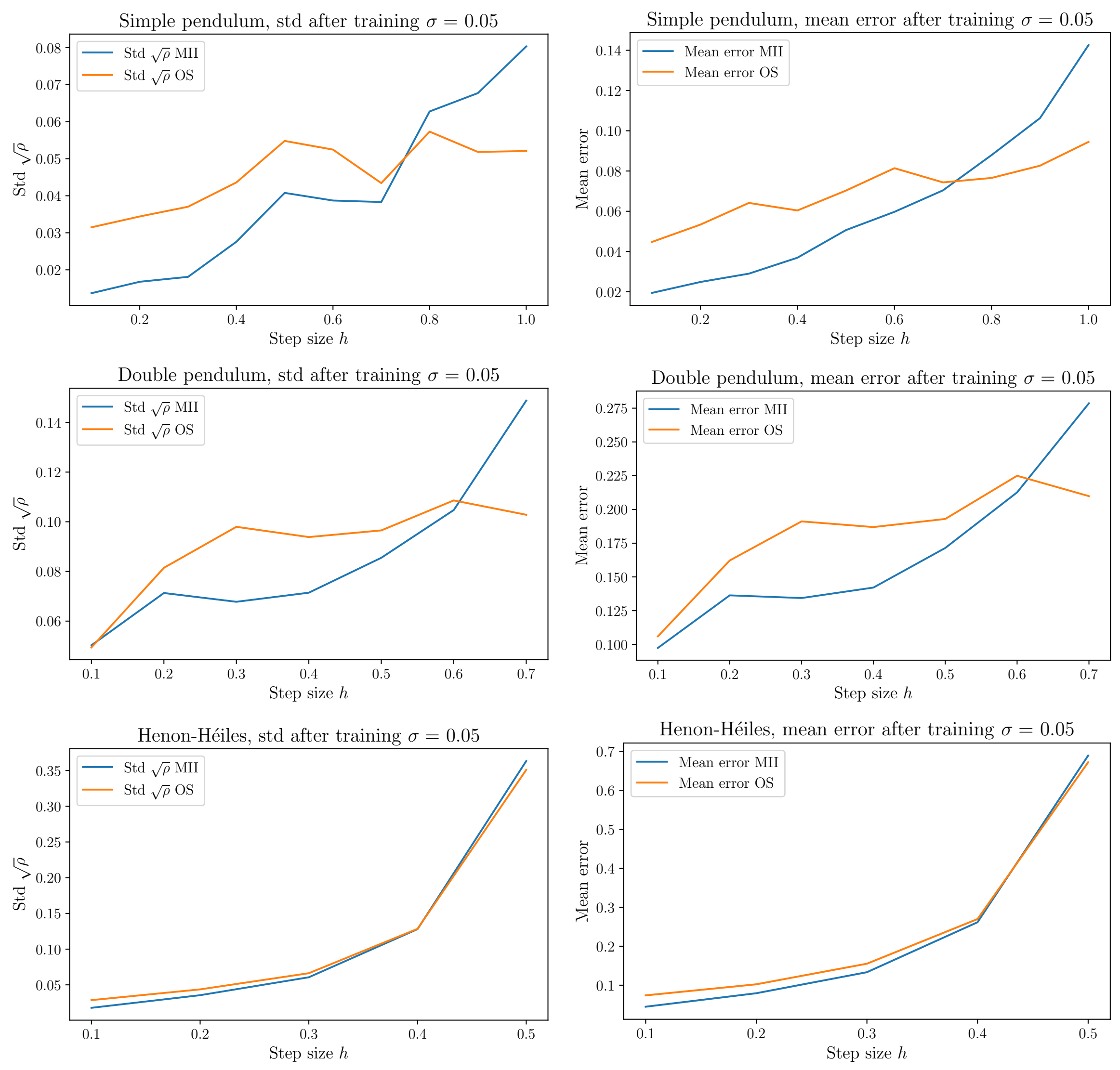


Figure 5.3: Results for the experiment following Algorithm 3. The standard deviation (left column) and mean (right column) of integration error for training multiple neural networks on realizations of noisy data, with decreasing step size h. Experiment done for the simple pendulum (top row), the double pendulum (mid row) and the Henon–Héiles ODE (lower row).

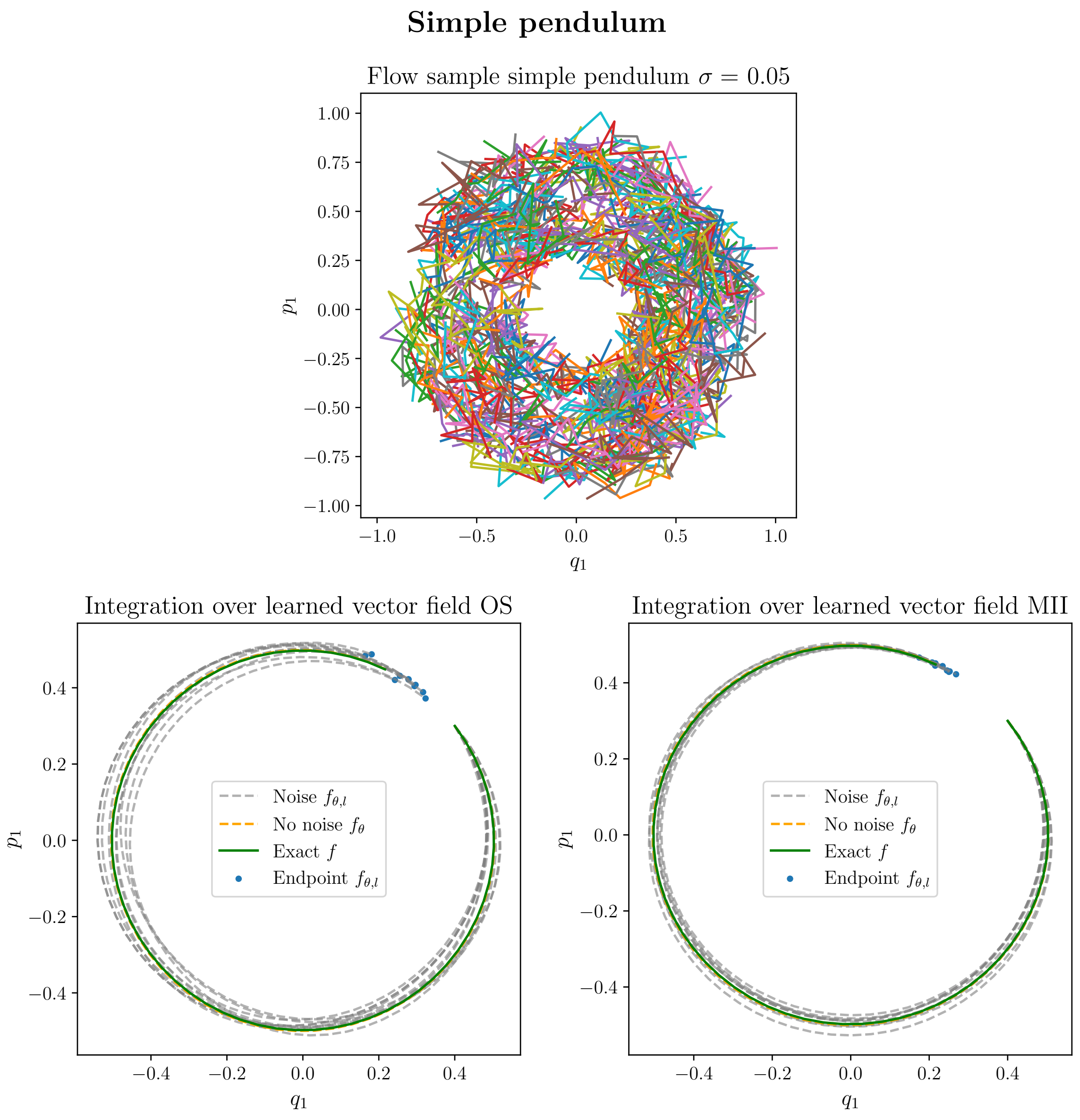


Figure 5.4: The upper plots display one sample of perturbed training data $\tilde{S}_l$ for the simple pendulum. The two lower plots display the numerical flow integrating over multiple learned vector fields trained on noisy data from the simple pendulum with $h_{\text{train}} = 0.1$ and $\sigma = 0.05$. The gray lines each represent one neural network $f_{\theta,l}$ that is trained on one realization of perturbed data $\tilde{S}_l$ with the midpoint metod as a one-step integrator (OS, left) and the mean inverse integrator (MII, right). The orange line is a neural network f_θ trained on data that is not perturbed and the green line is integration over the exact vector field.

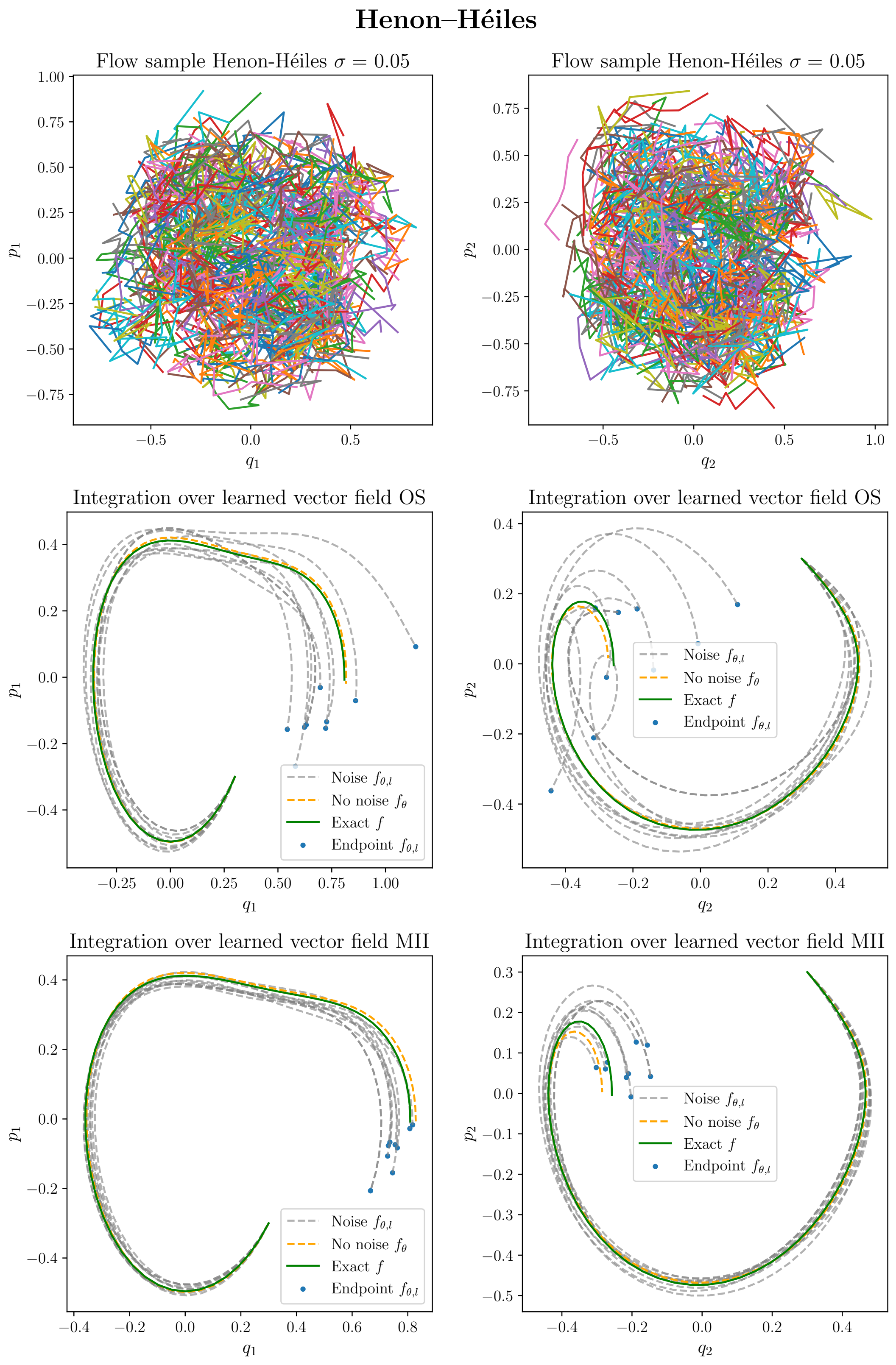


Figure 5.5: The upper plot displays two projections of one sample of perturbed training data $\tilde{S}_l$ for the Henon–Héiles ODE. The four lower plots display the numerical flow integrating over multiple learned vector fields trained on noisy data with $h_{\text{train}} = 0.1$ and $\sigma = 0.05$. The gray lines each represent one neural network $f_{\theta,l}$ that is trained on one realization of perturbed data $\tilde{S}_l$ with the midpoint method as a one-step integrator (OS, mid) and the mean inverse integrator (MII, lower). The orange line is a neural network f_θ trained on data that is not perturbed and the green line is integration over the exact vector field.

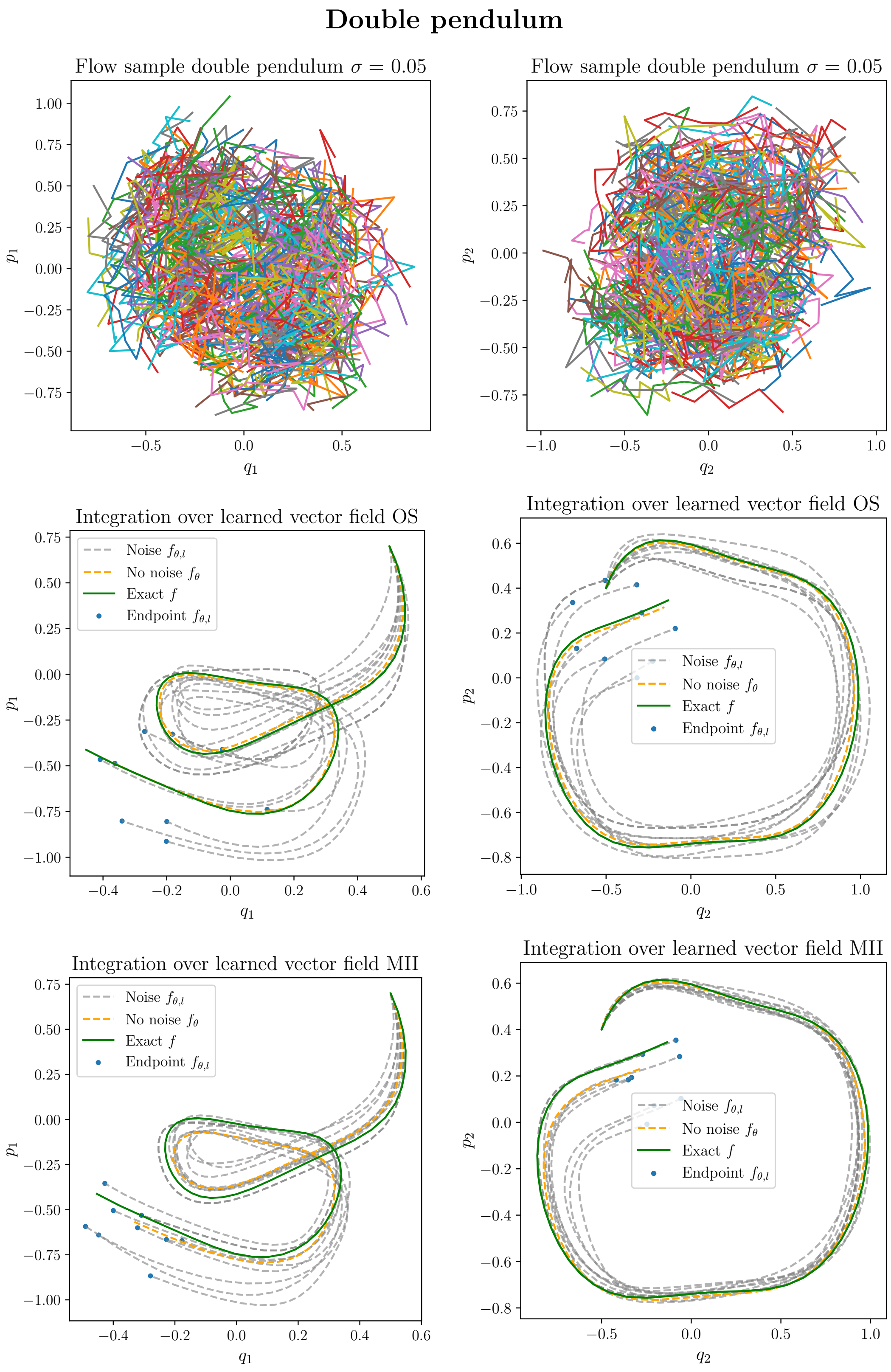


Figure 5.6: The upper plots display two projections of one sample of perturbed training data $\tilde{S}_l$ for the double pendulum. The four lower plots display the numerical flow integrating over multiple learned vector fields trained on noisy data with $h_{\text{train}} = 0.1$ and $\sigma = 0.05$. The gray lines each represent one neural network $f_{\theta,l}$ that is trained on one realization of perturbed data $\tilde{S}_l$ with the midpoint method as a one-step integrator (OS, mid) and the mean inverse integrator (MII, lower). The orange line is a neural network f_θ trained on data that is not perturbed and the green line is integration over the exact vector field.

5.4 Comparative testing of integrators

In order to compare the different integration methods introduced in the thesis, numerical experiments will be conducted for both the setting of a forward ODE problem, in addition to inverse Hamiltonian problems.

To verify the order of the integrator, the global integration error will be computed for decreasing step sizes. Let $\{h_i, N_i\}_{i=1}^{L}$ be pairs of step sizes $h_i > 0$ and $N_i + 1$ be the number of time steps such that $h_i \cdot N_i = T$ is the constant end time $T > 0$ where we evaluate the error. The error e_i for an integration method $\Phi_{h,f}$ is found by

$$e_i = \|y_{N_i} - y(T)\| \sim h_i^p,$$

for $i = 1, \dots, L$. Furthermore, the time used by the integration method to compute y_{N_i}, or the running time, by $\Delta t_c = t_{\text{end}} - t_{\text{start}}$, will be recorded in order to compare the computational efficiency of the different methods, when they are used to solve the forward problem. Finally, since we are studying Hamiltonian problems, we will measure if the Hamiltonian $H(y)$ is preserved along the solution trajectory y_i computed by the numerical integrator. This will be computed by

$$e_i^H = |H(y_0) - H(y_i)|,$$

for $i = 1, \dots, N$. Results on order, preservation of Hamiltonian and running time for integrating the Henon–Héiles Hamiltonian system, could be found in Figure 5.7.

For testing the accuracy of the integrators used in training, we will compute the mean of the global error over $\tilde{M}$ initial values, when integrating over a learned vector field f_θ that is trained on a flow sample with a step size h. Letting $y_{\tilde{N}}^{\theta}$ denote the point found when integrating $\tilde{N}$ steps over the learned vector field f_θ, the global error $\tilde{N}$ steps from initial value y_0 could be found by

$$e(f_\theta, y_0) = \|y_{\tilde{N}}^{\theta} - y(t_{\tilde{N}})\|,$$

and let us denote $e(f_\theta)$ as the mean error over $\tilde{M}$ different initial values.

Algorithm 4 Testing error of learned vector field for decreasing h

```
Require: Integrator for training $\Phi_{h,f}$ and trajectories $S_N^k = \{y_i\}_{i=0}^N$ for decreasing
  step sizes $h_k$
  for $k = 1, \dots, K$ do
      $h \leftarrow h^k$
      $S \leftarrow S_N^k$
      $\tilde{N} \leftarrow N_{\text{test}}^k$
      Train neural network $\min_\theta \mathcal{L}(\Phi_{h,f_\theta}, S, f_\theta)$
      Compute mean $e_k(f_\theta)$ integrating $\Phi_{h,f_\theta}$ over $\tilde{M}$ initial values, $\tilde{N}$ steps.
  end for
```

In this case, we let $h_k = \{0.5,\ 0.4,\ 0.3,\ 0.2,\ 0.1\}$, $N = 4$ and training on $M = 1000$ different trajectories with $r = 0.9$. The neural networks are trained for $n_{\text{epochs}} = 50$, computing the mean error over $\tilde{M} = 10$ different initial values for the simple and double pendulum. The results are found in Figure 5.8, in addition, the time it took to run the training using the different integrators could be studied in Figure 5.9.

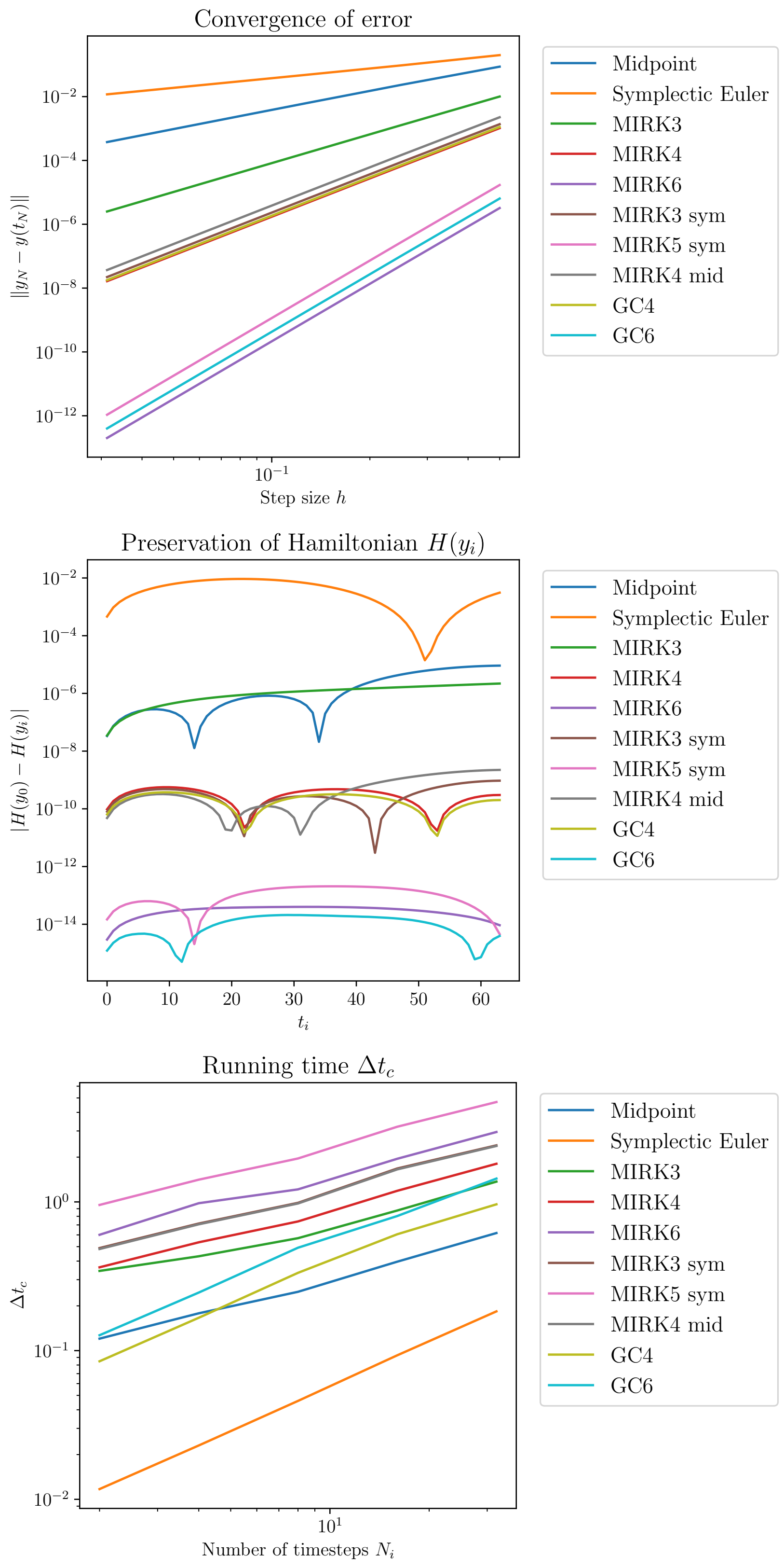


Figure 5.7: Testing integrators for the forward problem (known vector field). The plots present convergence of integration error e_i (top), the preservation of the Hamiltonian e_i^H (mid) and the running time Δt_c in seconds (bottom), for several integrators for the Henon–Héiles ODE.

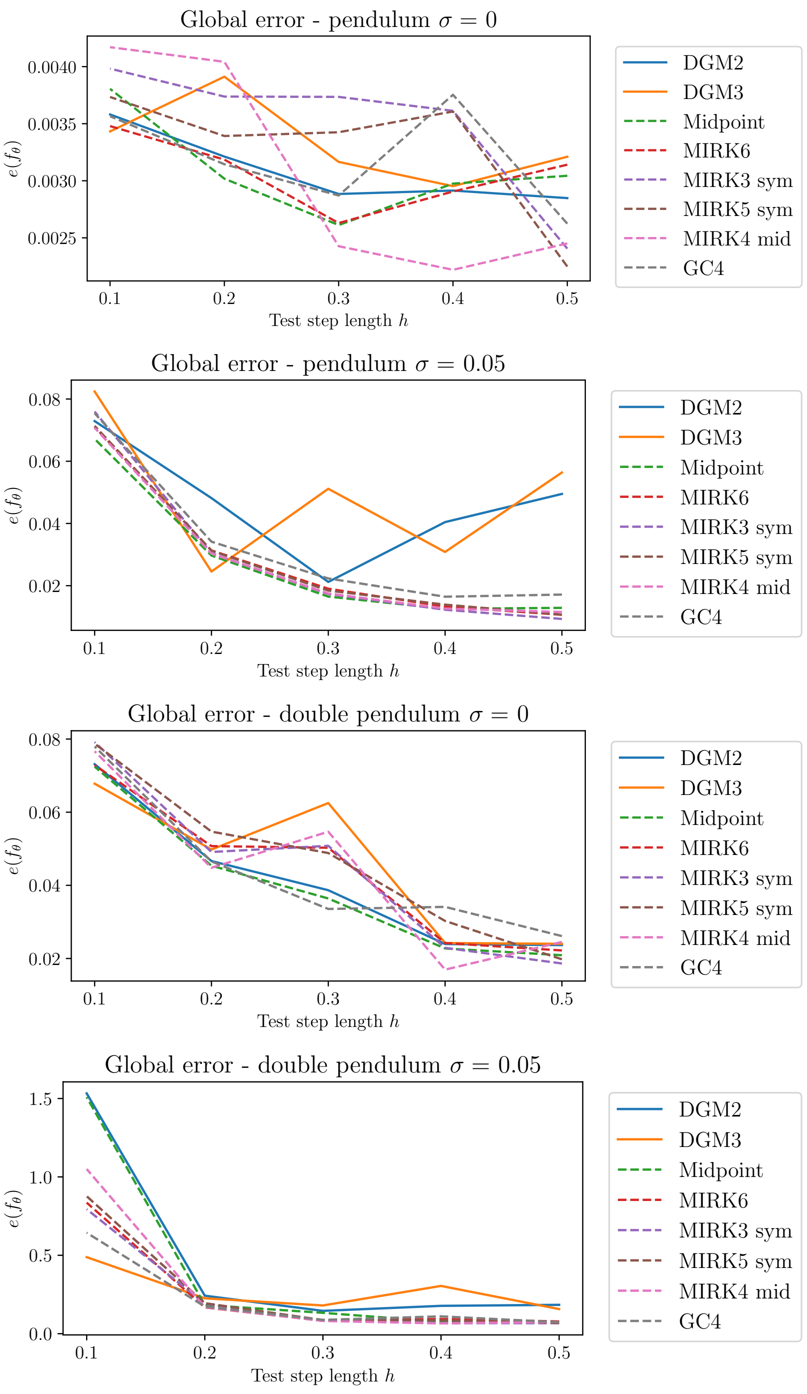


Figure 5.8: Results for the experiment following Algorithm 4. The error in the numerical flow $e(f_\theta)$ is computed after training neural networks with a range of integrators. The experiment done on the simple pendulum, double pendulum, with no noise in data (no. 1 and 3 from top) and with noise $\sigma = 0.05$ (no. 2 and 4 from top).

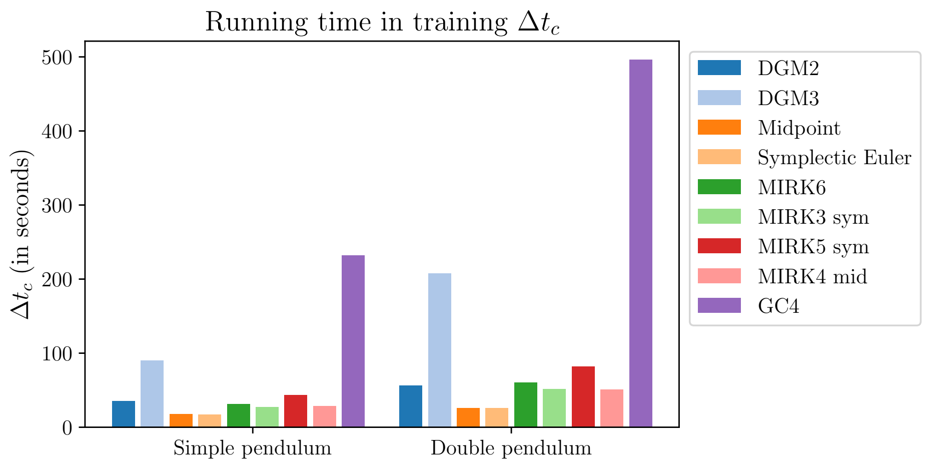


Figure 5.9: Running time (in seconds) when the numerical integrators are used to solve the inverse problem with data from the simple pendulum (left group) and the double pendulum (right group), or to train a neural network to represent the vector field.

Chapter 6

Discussion

This thesis aims at exploring the space of numerical integrators that could be used for solving inverse problems for ODEs and Hamiltonian ODEs in particular. By identifying that MIRK methods are explicit for inverse problems, we argue how this large class of numerical integrators could be efficiently used in solving the optimization problem over neural network parameters. One key question is thus how to make an informed decision about which MIRK method or in general which inverse explicit method to choose. Furthermore, in some cases it might be worthwhile to solve the non-linear equations required by an integration method that is not inverse explicit, such as GC4 or GC6. The question is then in which cases should this be done.

Comparative testing of integrators

The numerical experiment in Section 5.4 aims at giving answers to these questions. Studying Figure 5.8 the following could be noted:

- It is difficult to draw any conclusion on the experiments with noisy data ($\sigma = 0.05$). It could be that the error introduced from the normally distributed perturbations make the order of the integration method used in training, less significant.
- The *MIRK4 mid* method has relatively low error for large step sizes when there is not any noise in the data.
- The discrete gradient methods have relatively better performance for smallest step size, ($h = 0.1$).

However, all these remarks should only be considered as steppingstones for generating hypotheses for further experiments, and not as conclusive results. Furthermore, it should be noted that we exclusively study Hamiltonian systems in this thesis. Hence, it could be that geometric properties are more important in this setting, than order. Consider for instance Eidnes (2022), which study a port-Hamiltonian system and observe improved accuracy with increasing order (and symmetry) of the methods.

The results of using the presented Runge–Kutta methods for solving the forward problem of Henon-Héiles are found in Figure 5.7. Here, both GC4 and GC6 has lower running time than the MIRK methods, even though the system of equations has dimension sm in comparison to dimension m for the MIRK methods. However, when the integrators are used in training, the running time for GC4 is considerably longer since it requires backpropagation of gradients through the fixed point iterations considering Figure 5.9. Studying the error in the preservation of the Hamiltonian $H(y)$ and the convergence of the integration error, the MIRK4 method and GC4 has a performance that is more similar to each other than to *MIRK4 mid*, which was derived by modifying GC4. It could be a subject for future research to investigate whether this is due to some unknown shared features of MIRK4 and GC4.

Sensitivity to noise before and after training

The numerical experiments done in Section 5.2 and 5.3 compare how the mean inverse integrator and a regular one-step integration method differ in their sensitivity to noise in the data. This is first studied by assuming that the vector field is known and then examining how the two different approaches impact the distribution of the optimization target.

Then, several neural networks were trained on perturbed flow samples. By studying the distribution of points in the numerical flow obtained by integrating over these neural networks, we measured how noise was propagated by different integrators in the training. Consider first the experiments studying the variance in the optimization target:

- The covariance approximation for larger step size h presented in Theorem 4.14 corresponds well to the covariance obtained by sampling (when measuring the spectral radius) which could be studied in Figure 5.1 and 5.2.
- The approximation assuming small step size h, presented in Theorem 4.13, seems to be reasonable when $h < 0.1$ for all experiments.
- The double pendulum and Henon–Héiles are chaotic systems and more sensitive to noise. For instance, it is reasonable to assume that $\|f'(y)\|$ is larger for the double than for the simple pendulum. This could explain what is seen in Figure 5.1 and 5.2, where the standard deviation is much larger for the chaotic systems than for the simple pendulum. For the chaotic systems, it is clear that a significantly smaller step size h is needed for the mean inverse integrator to be expected to reduce the variance in the optimization target, compared to the simple pendulum.
- Studying the plots in the left column of Figure 5.1, the standard deviation measured by $\sqrt{\rho}$ for the mean inverse integrator has a significantly more convex shape, than the one-step method. This means that the approximations on the beginning and end of the trajectory will have larger variance than the points in the center of the trajectory. One way to (potentially) reduce the variance in the beginning and the end of the trajectory could be to reduce the number of different numerical integration trajectories included in the mean inverse integrator.

Secondly, the following could be noted about the experiment measuring the distribution of points in the flow after training neural networks on perturbed data.

- The mean inverse integrator yields lower mean error and standard deviation for sufficiently small step size h for all experiments found in Figure 5.3. However, when the step size is large, the mean inverse integrator has larger error and standard deviation. Hence, the performance of this method is highly dependent on the regularity of the underlying ODE and the sampling frequency or the step size h. The chaotic systems require the step size to be smaller, compared to the simple pendulum problem, in order to get a reduction in error and standard deviation.
- This is perhaps more clearly illustrated in the figures 5.4, 5.5 and 5.6, where a small error early in the trajectory yields a much larger deviation at a later point in time, for the Henon–Héiles system and the double pendulum, than for the simple pendulum ODE.

As a general remark, experiments involving neural networks are stochastic. This is due to the random initialization of the parameters θ, meaning the weight matrices W_i and bias vectors b_i. Since the L-BFGS optimization algorithm is used instead of stochastic gradient descent or Adams, the training procedure itself is less stochastic. Ideally the experiments involving neural network training should have been re-run multiple times with different initialization to ensure robust results and to quantify just how stochastic the experimental results are.

Chapter 7

Conclusion

By describing the inverse explicit property of numerical integrators and identifying the class of mono-implicit Runge–Kutta methods as inverse explicit, the toolbox for inverse ODE problems has been expanded. Unfortunately, as proved in this thesis, the highest order of a symplectic, MIRK method is $p = 2$, which could be limiting for particular types of problems. However, as mentioned in the discussion following the numerical experiments, it is challenging to demonstrate any consistent results that could guide the choice of numerical integrator for a specific inverse ODE problem. It is hard to show consistently to what extent symmetry, symplecticity or even the order of the integration method gives improved training for the problems studied in this thesis.

The mean inverse integrator, on the other hand, is introduced as a method demonstrating that a more radical rethinking of how a numerical integrator could be used, could yield an optimization problem less sensitive to perturbations in the data. This, again could lead to learning a more accurate approximation of the vector field, but only for sufficiently regular problems and for sufficiently small step sizes in time h. Hence, this method should be used with some caution.

The increasing amount of data produced by various types of sensors and the growing number of autonomous systems calls for robust algorithms for prediction and control of dynamical systems. The energy-based Hamiltonian and Lagrangian equations from classical mechanics paired with methods from deep learning could thus be helpful. However, it is important to pay proper attention to how these equations are discretized in time and how the optimization problems for solving the inverse problems are structured. This thesis provides some guidance, as well as raising a series of new questions on how to solve these problems in the best way.

Multiple directions could be followed for expanding on this research. More thorough analysis could be done to describe the properties of the mean inverse integrator. Consider for instance deriving an expression for the integration error by accumulating local error from taking compositions of integration steps, or finding the optimal weights for summing over the different numerical trajectories. More sophisticated numerical experiments on a greater range of dynamical system could perhaps yield more consistent results on the question of which integration method to choose for a specific inverse ODE problem. Furthermore, trying to learn the dynamics of more intricate and higher dimensional systems, such as the oscillatory Fermi–Pasta–Ulam–Tsingou problem (Hairer et al., 2006, Ch. I.5.1), would be an interesting endeavor.

Bibliography

Bai, S., Kolter, J.Z., Koltun, V., 2019. Deep equilibrium models. Advances in Neural Information Processing Systems 32.

Benning, M., Celledoni, E., Ehrhardt, M.J., Owren, B., Schönlieb, C.B., 2019. Deep learning as optimal control problems: models and numerical methods. Journal of Computational Dynamics 6, 171–198.

Burrage, K., Chipman, F., Muir, P.H., 1994. Order results for mono-implicit Runge–Kutta methods. SIAM journal on numerical analysis 31, 876–891.

Celledoni, E., Leone, A., Murari, D., Owren, B., 2022. Learning Hamiltonians of constrained mechanical systems. arXiv preprint arXiv:2201.13254 .

Chartier, P., Hairer, E., Vilmart, G., 2007. Numerical integrators based on modified differential equations. Mathematics of computation 76, 1941–1953.

Chen, R., Tao, M., 2021. Data-driven prediction of general Hamiltonian dynamics via learning exactly-symplectic maps, in: International Conference on Machine Learning, PMLR. pp. 1717–1727.

Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K., 2018. Neural ordinary differential equations. Advances in neural information processing systems 31.

Chen, Y., Matsubara, T., Yaguchi, T., 2021. Neural Symplectic Form: Learning Hamiltonian Equations on General Coordinate Systems. Advances in Neural Information Processing Systems 34.

Chen, Z., Zhang, J., Arjovsky, M., Bottou, L., 2019. Symplectic recurrent neural networks. arXiv preprint arXiv:1909.13334 .

David, M., Méhats, F., 2021. Symplectic Learning for Hamiltonian Neural Networks. arXiv preprint arXiv:2106.11753 .

Dormand, J., Prince, P., 1980. A family of embedded Runge–Kutta formulae. Journal of Computational and Applied Mathematics 6, 19–26. URL: https://www.sciencedirect.com/science/article/pii/0771050X80900133, doi:https://doi.org/10.1016/0771-050X(80)90013-3.

Duong, T., Atanasov, N., 2021. Hamiltonian-based neural ODE networks on the SE (3) manifold for dynamics learning and control. arXiv preprint arXiv:2106.12782 .

Eidnes, S., 2022. Order theory for discrete gradient methods. BIT Numerical Mathematics , 1–49.

Eidnes, S., Stasik, A.J., Sterud, C., Bøhn, E., Riemer-Sørensen, S., 2022. Port-Hamiltonian Neural Networks with State Dependent Ports URL: https://arxiv.org/abs/2206.02660, doi:10.48550/ARXIV.2206.02660.

Elkabetz, O., Cohen, N., 2021. Continuous vs. discrete optimization of deep neural networks. Advances in Neural Information Processing Systems 34.

Enright, W.H., Muir, P.H., 1986. Efficient classes of Runge–Kutta methods for two-point boundary value problems. Computing 37, 315–334.

Finzi, M., Wang, K.A., Wilson, A.G., 2020. Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. arXiv preprint arXiv:2010.13581 .

Goldstein, H., Poole, C., Safko, J., 2001. Classical Mechanics. 3 ed., Addison Wesley.

Gonzalez, O., 1996. Time integration and discrete Hamiltonian systems. Journal of Nonlinear Science 6, 449–467.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Greydanus, S., Dzamba, M., Yosinski, J., 2019. Hamiltonian Neural Networks. CoRR abs/1906.01563. URL: http://arxiv.org/abs/1906.01563, arXiv:1906.01563.

Haber, E., Ruthotto, L., 2017. Stable architectures for deep neural networks. Inverse problems 34, 014004.

Hairer, E., Lubich, C., Wanner, G., 2006. Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations; 2nd ed. Springer, Dordrecht. doi:10.1007/3-540-30666-8.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585, 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2, doi:10.1038/s41586-020-2649-2.

Jin, P., Zhang, Z., Zhu, A., Tang, Y., Karniadakis, G.E., 2020. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. Neural Networks 132, 166–179.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Le, Q.V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Ng, A.Y., 2011. On optimization methods for deep learning, in: ICML.

Leimkuhler, B., Reich, S., 2005. Simulating Hamiltonian Dynamics. Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press. doi:10.1017/CBO9780511614118.

Matsubara, T., Ishikawa, A., Yaguchi, T., 2019. Deep energy-based modeling of discrete-time physics. arXiv preprint arXiv:1905.08604 .

McLachlan, R.I., Quispel, G.R.W., Robidoux, N., 1999. Geometric integration using discrete gradients. Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 357, 1021–1045.

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A., 2017. SymPy: symbolic computing in Python. PeerJ Computer Science 3, e103. URL: https://doi.org/10.7717/peerj-cs.103, doi:10.7717/peerj-cs.103.

Nocedal, J., Wright, S.J., 1999. Numerical optimization. Springer.

Noren, H., 2022. Specialization project in industrial mathematics: Preserving invariants in inverse problems. Unpublished .

Offen, C., Ober-Blöbaum, S., 2022. Symplectic integration of learned Hamiltonian systems. Chaos: An Interdisciplinary Journal of Nonlinear Science 32, 013122.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. PyTorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32, 8026–8037.

Poli, M., Massaroli, S., Yamashita, A., Asama, H., Park, J., 2020. Hypersolvers: Toward fast continuous-depth models. Advances in Neural Information Processing Systems 33, 21105–21117.

Quarteroni, A., Sacco, R., Saleri, F., 2010. Numerical mathematics. volume 37. Springer Science & Business Media.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., Edelman, A., 2020. Universal differential equations for scientific machine learning URL: http://dx.doi.org/10.21203/RS.3.RS-55125/V1, doi:10.21203/rs.3.rs-55125/v1.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics 378, 686–707.

Sanz-Serna, J.M., Calvo, M.P., 2018. Numerical Hamiltonian problems. Courier Dover Publications.

Shi, B., Du, S.S., Su, W., Jordan, M.I., 2019. Acceleration via symplectic discretization of high-resolution differential equations. Advances in Neural Information Processing Systems 32.

Sundklakk, H.S., 2015. A Library for Computing with Trees and B-Series. Master's thesis. NTNU.

Van Der Schaft, A., Jeltsema, D., 2014. Port-Hamiltonian systems theory: An introductory overview. Foundations and Trends in Systems and Control 1, 173–378.

Van Loan, C.F., 2000. The ubiquitous kronecker product. Journal of computational and applied mathematics 123, 85–100.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17, 261–272. doi:10.1038/s41592-019-0686-2.

Wanner, G., Hairer, E., 1996. Solving ordinary differential equations II. volume 375. Springer Berlin Heidelberg.

Weinan, E., 2017. A proposal on machine learning via dynamical systems. Communications in Mathematics and Statistics 1, 1–11.

Williams, R.J., Zipser, D., 1989. A learning algorithm for continually running fully recurrent neural networks. Neural computation 1, 270–280.

Yoshida, H., 1990. Construction of higher order symplectic integrators. Physics letters A 150, 262–268.

Zhong, Y.D., Dey, B., Chakraborty, A., 2020. Dissipative SymODEN: Encoding Hamiltonian dynamics with dissipation and control into deep learning. arXiv preprint arXiv:2002.08860 .

Zhu, A., Jin, P., Tang, Y., 2020. Deep Hamiltonian networks based on symplectic integrators. arXiv preprint arXiv:2004.13830 .

NTNU
Norwegian University of
Science and Technology

SINTEF