Doctoral theses at NTNU, 2024:240

Andrea Leone

Data-driven and geometric numerical methods for mechanical systems

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of Science and Technology

Andrea Leone

Data-driven and geometric numerical methods for mechanical systems

Thesis for the Degree of Philosophiae Doctor

Trondheim, June 2024

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

NTNU

Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-326-8072-6 (printed ver.)
ISBN 978-82-326-8071-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:240

Printed by NTNU Grafisk senter

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of *Philosophiae Doctor* (Ph.D.) at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. It marks the conclusion of around four years of work carried out at the Department of Mathematical Sciences (IMF), with one year dedicated to teaching assistance and approximately a semester to coursework, and covers a series of projects in which I was involved throughout this time.

First of all, I would like to express my deepest gratitude to my supervisor, Elena Celledoni, for her constant guidance and the insightful comments and suggestions, as well as patience and motivation during times of low morale and self-confidence. Likewise, I am extremely grateful to Brynjulf Owren for his advice and support. In these years, I have gained valuable insights and skills from their vast knowledge and expertise. I have also been fortunate to work closely with many awesome people within their research group, and I extend my sincere thanks to all of them for the fruitful academic experiences and constructive discussions. Hoping not to disappoint anyone, I will avoid providing an exhaustive list and will only mention Davide Murari and Ergys Çokaj, who have been there from the very beginning. Thank you for being excellent companions during our Ph.D. journey. I am also much obliged to Martin Arnold and Per Thomas Moe for their generous dedication of time, support, and stimulating discussions as members of my supervisory committee.

This doctoral research was conducted as part of the European Training Network THREAD (Marie Skłodowska-Curie grant agreement No. 860124), where I served as Early Stage Researcher 4. The network has provided me with invaluable learning opportunities, personal growth, and meaningful friendships. I express my gratitude to Martin Arnold and the THREAD coordination team for their efficient management of this extensive network. I can barely imagine the tremendous effort required to lead such a large consortium, especially amidst the challenges posed by the COVID-19 pandemic, and I appreciate the dedication involved. Although it would be impossible to acknowledge each participant individually, I wish to specifically thank Sigrid Leyendecker, Olivier Brüls, and Joachim Linn for hosting me during my one-month research sec-

ondments. These experiences allowed me to become familiar with different research environments and to delve into further topics like discrete mechanics of beams, simulation of cables, and interpolation on Lie groups. Furthermore, were it not for THREAD, I would not have had the opportunity to spend a three-month secondment at TechnipFMC, within the Structural Analysis Engineering team led by Per Thomas Moe, to whom I am sincerely grateful. This was an incredible occasion to gain insight into industrial problem solving, particularly in the field of subsea engineering. It was exciting to collaborate with the local Ph.D. student Halvor S. Gustad on the analysis of sensor data from offshore operations.

As a fellow of the THREAD project and a member of Elena and Brynjulf's group, I have had the privilege of meeting and engaging with exceptional and committed scientists and researchers. I will always treasure experiences such as MaGIC, HB60, HFSS2023, and the THREAD's annual meetings.

Of course, I cannot refrain from expressing my affection for the other Ph.D. students in THREAD. I owe them beautiful memories and I am thankful for the wonderful moments we shared, both within and beyond the academic setting. By the same token, I am grateful to all the friends and colleagues at NTNU for the pleasant coffee breaks and lunches on the 13th floor, the dinners, the payday beers, and for all the cherished time spent together. Undertaking a Ph.D. at the Department of Mathematical Sciences at NTNU significantly influenced my academic journey. I believe it is an ideal environment for doctoral studies, and I would like to recognise the entire IMF, in particular the Head of Department Einer Rønquist and the administration.

A thorough list of people I wish to acknowledge might unintentionally omit someone. To all those who have been there for me, I express my heartfelt appreciation. Special thanks to all friends, either close by or far away, who supported me with meaningful conversations, sometimes enduring too long voice messages.

Last but certainly not least, I would like to thank my family for their unwavering love. To them I dedicate this thesis.

The past four years have been filled with excitement and challenges. The pursuit of a Ph.D. has definitely been a whirlwind of emotions, alternating between moments of happiness and satisfaction and periods of crisis and frustration. Nevertheless, what a remarkable journey!

Andrea Leone
Trondheim, March 12, 2024

Contents

Introduction

Many mechanical systems relevant to engineering applications can be modelled as multibody systems. These systems consist of a finite number of rigid and elastic bodies that undergo large displacements and rotations. For example, lumped mass systems are described as a combination of bars or links acting as connecting elements, concentrated masses accounting for inertia, springs that include stiffness, and dampers that dissipate energy [4, 44]. The configuration space of these systems is typically nonlinear, since it consists of combinations of rotations and translations, and is mathematically defined as a manifold. More involved examples are represented by flexible slender structures like beams, cables, and hoses. Cosserat rod theory (see [3] and references therein) provides a well-established model for the geometrically exact simulation of deformable rods. The configuration of a Cosserat rod is represented by a curve of centroids (or centre line) and the orientation of its cross sections, and thus evolves in a nonlinear space as well (see, e.g., [41, 57]). Although they are essential in high-performance engineering processes, the behaviour of these systems under real operating conditions exceeds the capabilities of the modelling tools used in the current product development cycles [1]. An important research area in this context focusses on developing sophisticated modelling techniques for accurate simulations while preserving fundamental geometric properties. In particular, nonlinear configuration spaces often have a Lie group structure, making it advantageous to apply principles of differential geometry in developing efficient numerical algorithms. The main methodology to construct numerical discretisations that can cope with high flexibility and large rotations is based on a finite-element space discretisation and geometric numerical integrators for time evolution. When it comes to finite-element discretisation, one of the main challenges is the accurate approximation of finite rotations. For this reason, methods for interpolation on manifolds (specifically the Lie groups of rotations *SO(3)* or of rigid motions *SE(3)*) are used instead of the usual piecewise-polynomial interpolation [6, 57].

Geometric numerical integration is a field of research concerning time integration of differential equations with underlying geometric structure, as, for example, the preservation of symplecticity for Hamiltonian systems, the preser-

vation of volume or of invariants [33]. In the context of this thesis, Lie group integrators, designed for solving ordinary differential equations on Lie groups and homogeneous manifolds [18], are the geometric numerical integrators of importance.

The traditional approach for the modelling and analysis of physical systems in the fields of science and engineering has mostly hinged on physics-based modelling. This involves formulating governing equations and developing efficient computational algorithms of proven consistency, stability and convergence. This approach has the advantage of being rooted in first principles, which ensure interpretability and generalisability, and can be supported by solid mathematical foundations. On the other hand, the abundance of observational data and the increasing power of modern computing infrastructures have fostered the growth of a new data-driven paradigm, in which machine learning algorithms play a pivotal role (see, e.g., [11] for a detailed overview of the latest advancements). This approach relies on a large amount of data to perform inference or prediction, and can be beneficial in situations where the underlying physics is not fully captured by the existing model [8], or when solving high-dimensional equations becomes challenging with traditional methods [31]. Nevertheless, several difficulties can arise in this scenario, which include potential concerns regarding the model's performance on previously unseen data, the limited interpretability of the results of most deep learning algorithms, and the lack of a proper mathematical theory to investigate stability and uncertainty. Numerous paths of research are currently being explored to address these issues. One approach involves incorporating physical laws into the learning process [25, 26, 29, 38, 53, 54], while others leverage ideas from dynamical systems, geometry, and optimisation theory to design and analyse neural networks [7, 10, 15, 16, 19, 22, 25, 32, 49, 58].

This thesis focusses on applications of structure-preserving numerical methods and data-driven techniques to mechanical problems. It is a collection of five papers, either published or in the submission process, each of which constitutes a chapter. We study prototypical examples that encompass the aspects and challenges mentioned above, aiming to illustrate the advantages and limitations of the different approaches. The objective is to develop suitable techniques that could be beneficial for the simulation of more complex systems, such as real-life systems featuring highly flexible slender structures. There are two main parts to this dissertation. The first one consists of papers 1, 2, and 3. This part considers Lie group integrators of the Runge–Kutta–Munte–Kaas (RKMK) and commutator free type and includes different numerical applications for Lagrangian and Hamiltonian systems evolving on manifolds [13, 14]. These methods are then employed in the data-driven modelling of Hamiltonian systems, providing an example in which deep learning algorithms are com-

bined with geometric numerical integrators on homogeneous manifolds [17]. The second part of the thesis includes papers 4 and 5, and is more focused on the use of fully data-driven procedures on two specific problems related to the fields of mechanical and subsea engineering. It is based on supervised machine learning techniques, in particular deep learning, for prediction and classification. Paper 4 is based on a collaboration with the Institute of Applied Dynamics at FAU Erlangen-Nürnberg, whereas paper 5 is the result of an internship at the industrial partner TechnipFMC in Lysaker. These works explore the potential of machine learning-assisted methods in the fields of computational mechanics [20] and engineering decision-making [23].

The remainder of this chapter introduces basic concepts and notation used in the thesis, deferring most of the details to the forthcoming papers. It does not aim to provide a comprehensive coverage of topics concerning differential geometry, geometric mechanics, and geometric numerical integration, for which readers are directed to the standard textbooks and excellent expositions such as [5, 33, 35, 43, 44, 50] and references therein. Finally, a concise overview of each article is provided.

1.1 Mechanical systems on manifolds

Let the configuration of a mechanical system be represented by a set of d generalised coordinates collected in the tuple $q = (q_1, \dots, q_d)$, with d corresponding to the degrees of freedom of the system. The set of all configurations defines the d-dimensional configuration space Q. Given a trajectory of the system described by the differentiable curve $q(t) : [t_0, t_N] \to Q$, the time derivative $\dot{q}(t)$ defines the velocity at time t, which belongs to the tangent space of Q at $q(t)$, that is, $\dot{q} \in T_q Q$. The state of the system is then determined by $(q, \dot{q}) \in TQ$, the tangent bundle of the configuration manifold, which represents the state space (velocity phase space) of the system.

In most of the thesis, we consider mechanical systems from a Lagrangian or Hamiltonian perspective. Lagrangian mechanics is based on the principle that motions of the systems are extremals of an action functional. The Lagrangian function $L : TQ \to \mathbb{R}$ is usually given by the difference of the kinetic energy and a potential energy accounting for elastic deformation and possible external loading. Hamilton's variational principle states that the action functional, defined as the integral of the Lagrangian along a motion of the system over a fixed time interval, is stationary to first order. This requirement leads to the Euler-Lagrange equations of motion, that govern the evolution of the physical system. Let us consider a first order Lagrangian

$$L : TQ \to \mathbb{R}, \quad L(q, \dot{q}) = \frac{1}{2} \dot{q}^T M(q) \dot{q} - U(q) \tag{1.1.1}$$

defined by the difference between a kinetic energy, usually expressed as a positive-definite quadratic form, and a potential energy that we assume be dependent solely on the configuration variables. We can perform a change of coordinates $(q, \dot{q}) \to (q, p)$ by introducing the conjugate momentum $p \in T_q^*(Q)$, lying in the cotangent space of Q at q, via the Legendre transform

$$\mathbb{F}L : TQ \to T^*Q, \quad (q, \dot{q}) \mapsto (q, p) := \left(q, \frac{\partial L}{\partial \dot{q}}(q, \dot{q}) \right), \tag{1.1.2}$$

which we assume to be a global isomorphism between the tangent bundle and the cotangent bundle of the configuration manifold Q [1]. As a result, introducing the Hamiltonian function on the cotangent bundle of Q (the phase space of the system) as

$$H : \mathrm{T}^*Q \to \mathbb{R}, \quad H(q, p) = p \cdot \dot{q} - L\left(q, \dot{q}\right)\Big|_{(q,\dot{q})=\mathbb{F}L^{-1}(q,p)}, \tag{1.1.3}$$

one can obtain the Hamilton's equations of motion. We notice that, in the case of the Lagrangian in (1.1.1), we have $p = M(q)\dot{q}$ and

$$\begin{aligned} H(p, q) &= p \cdot M(q)^{-1} p - L\left(q, M(q)^{-1} p\right) \\ &= p \cdot M(q)^{-1} p - \frac{1}{2} p^T M(q)^{-1} p + U(q) = \frac{1}{2} p^T M(q)^{-1} p + U(q), \end{aligned} \tag{1.1.4}$$

thus the Hamiltonian is the sum of the kinetic and potential energy of the mechanical system.

The configuration space is in general a differentiable manifold that arises from physical constraints imposed on the motion of the system. In particular, we consider holonomic constraints defined by the function

$$g : \mathbb{R}^n \to \mathbb{R}^m, \quad g(q) = 0 \in \mathbb{R}^m, \quad m < n. \tag{1.1.5}$$

We assume that $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$, are scalar differentiable functions with linearly independent gradients, so that

$$Q = g^{-1}(0) = \{q \in \mathbb{R}^n \mid g(q) = 0\} \tag{1.1.6}$$

defines the constraint manifold of dimension $d := n - m$ embedded in $\mathbb{R}^n$. The tangent bundle

$$\begin{aligned} TQ &= \{(q, \dot{q}) \in \mathbb{R}^n \times \mathbb{R}^n \mid q \in Q,\ \dot{q} \in T_q Q\} \\ &= \{(q, \dot{q}) \in \mathbb{R}^n \times \mathbb{R}^n \mid g(q) = 0,\ G(q)\dot{q} = 0\}, \end{aligned} \tag{1.1.7}$$

[1] This means that the Legendre transform is an invertible map, so that it defines a valid change of variables. In this case, the Lagrangian is said to be hyperregular [35, 44, 50], and this is a key assumption to establish the upcoming equivalence between the Lagrangian and Hamiltonian equations of motion.

where $G(q)$ is the Jacobian matrix of g, is a $2d$-dimensional smooth manifold, and an analogue result holds for the cotangent bundle of the constrained configuration manifold [44, sec. 1.2].

In the following, we always assume that the configuration manifold is an embedded submanifold of a finite-dimensional vector space $\mathbb{R}^n$, called the ambient space. This enables us to extend the system under consideration to the embedding vector space and to represent a point $q \in Q$ as a vector in $\mathbb{R}^n$, provided that the equality conditions (1.1.5) are satisfied [44, p. 12]. Alternatively, the manifold can be defined locally by a parametrisation (see, e.g., [35, sec. 2.1] or [33, sec. IV.5.1]), so that the constrained system can be fully described by a set of d local coordinates or parameters.

A common approach to model constrained mechanical systems is to introduce Lagrange multipliers to couple constraints to the Lagrangian function, thus confining the evolution of the system on the tangent bundle of the constraint manifold. The resulting augmented Lagrangian is used to derive constrained Euler-Lagrange equations that assume the form of differential-algebraic equations [4]. An example of this approach is shown in Section 5.2.

In general, expressing constraints in terms of manifolds provides a method to encode them directly, without resorting to a formulation based on a set of minimal local coordinates, which typically encounter singularities, or on the redundant coordinates of the embedding space along with Lagrange multipliers. This approach conveniently exploits the tools of differential geometry and results in a global, coordinate-free description of the system valid on the entire configuration manifold. In particular, as shown in [44], a global formulation of the Euler–Lagrange equations or Hamilton's equations can be derived by variational methods, where variations are restricted to the geometry of the configuration manifold by means of an orthogonal projection operator mapping $\mathbb{R}^n$ onto the tangent space $\mathrm{T}_q Q$, for each $q \in Q$. This geometric framework is adopted in various examples in the first three papers.

This section concludes with examples of manifolds that are significant in the context of Lagrangian and Hamiltonian systems, such as Lie groups and homogeneous manifolds, that are particularly relevant for this thesis.

A Lie group G is both a differentiable manifold and an algebraic group equipped with a smooth binary operation $G \times G \to G$ and a smooth inversion map (see, e.g, [36, Definition 2.8]). We consider, in particular, matrix Lie groups [2], with the matrix multiplication as group operation. One of the most representative matrix Lie groups is the group $GL(n)$ of invertible $n \times n$ matrices,

[2] We focus on matrix Lie groups with real entries, that are submanifolds of the space $M(n, \mathbb{R})$ of $n \times n$ real matrices. The latter is a linear space and can be identified with $\mathbb{R}^{n \times n} = \mathbb{R}^{n^2}$ [35, sec. 5.1].

called the general linear group,

$$GL(n) = \left\{ A \in M(n,\mathbb{R}) \mid \det A \neq 0 \right\}. \tag{1.1.8}$$

It can be shown that it is an n^2-dimensional submanifold of the space of real matrices $M(n,\mathbb{R})$ [35, p. 168]. The special linear group $SL(n)$ is a subgroup of $GL(n)$ composed of matrices with determinant 1, and has dimension n^2-1 (in fact, we are adding a scalar constraint with respect to $GL(n)$). The group $O(n)$ of orthogonal $n \times n$ matrices is a $n(n-1)/2$-dimensional Lie group, and the same applies for the special orthogonal group $SO(n)$ of all $n \times n$ orthogonal matrices with determinant 1 [35, sec. 5.1], $SO(n) = O(n) \cap SL(n)$, that is,

$$SO(n) = \left\{ A \in GL(n) \mid A^\top A = AA^\top = I, \det A = 1 \right\}, \tag{1.1.9}$$

with I the $n \times n$ identity matrix. The group of proper rotations $SO(3)$ of $\mathbb{R}^3$ is of significant importance in practical applications, as is the Lie group of homogeneous transformations in three dimensions, denoted by $SE(3)$. The latter is called the special Euclidean group, and its elements (R, v) can be represented as 4×4 matrices of the form

$$(R, v) = \begin{bmatrix} R & v \\ 0 & 1 \end{bmatrix}, \tag{1.1.10}$$

where $R \in SO(3)$, $v \in \mathbb{R}^3$, 0 is a row vector in $\mathbb{R}^3$ and 1 is a real number. Given a position vector $w \in \mathbb{R}^3$, we have

$$\begin{bmatrix} R & v \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w \\ 1 \end{bmatrix} = \begin{bmatrix} Rw + v \\ 1 \end{bmatrix}, \tag{1.1.11}$$

which shows that a group element (R, v) can be used to rotate and translate a vector in the three-dimensional space, that is, $SE(3)$ describes the rigid motions of $\mathbb{R}^3$. More formally, the special Euclidean group is defined as the semidirect product of $SO(3)$ and $\mathbb{R}^3$,

$$SE(3) = SO(3) \ltimes \mathbb{R}^3. \tag{1.1.12}$$

This means that, as a set, it is defined by pairs $(R, v) \in SO(3) \times \mathbb{R}^3$, but the group operation is defined as $(R_2, v_2)(R_1, v_1) = (R_2 R_1, v_2 + R_2 v_1)$, as can be easily verified using the representation in (1.1.10) [3]. It is possible to show that $SE(3)$ is a six-dimensional manifold embedded in $GL(4)$ [44, p. 24].

The notion of group action is fundamental for the upcoming discussion. We denote by $\mathcal{M}$ a generic differentiable manifold. A (left) group action is a map $\psi : G \times \mathcal{M} \to \mathcal{M}$ such that

[3] The set $SO(3) \times \mathbb{R}^3$ with the natural composition $(R_2, v_2)(R_1, v_1) = (R_2 R_1, v_2 + v_1)$ is instead referred to as the direct product of $SO(3)$ and $\mathbb{R}^3$ [44, p. 9].

- $\psi(h, \psi(g, y)) = \psi(hg, y)$ for all $h, g \in \mathcal{G}$ and $y \in \mathcal{M}$,
- $\psi(e, y) = y$ for all $y \in \mathcal{M}$, with e the identity element of G.

Every Lie group acts on itself by left multiplication. Another key concept is that of transitive action, which allows us to transit from any location $x \in \mathcal{M}$ to any other location y in $\mathcal{M}$. More specifically, the group action ψ is said to be transitive provided that for every $x, y \in \mathcal{M}$ there exists $g \in \mathcal{G}$ such that $y = \psi(g, x)$. In this case, the manifold $\mathcal{M}$ is a homogeneous manifold, acted upon by G (see, e.g., [37]). An example that will be used extensively in this thesis is the unit sphere in the three-dimensional space, that is acted upon transitively by $SO(3)$.

Another example of interest in the context of computational mechanics is the group of unit quaternions, that can provide a parametrisation of the rotational degrees of freedom of a system (for example, they are used in [41] to specify the orientation of the cross sections in a geometrically exact beam model). Furthermore, products of the aforementioned manifolds are often found as configuration manifolds of mechanical systems, as shown in some of the examples provided in paper 1.

1.2 Structure-preserving numerical methods

Mechanical systems are usually characterised by invariant quantities of significant physical meanings, like energy for conservative systems or total momentum for systems with Lagrangians that are invariant under spatial translation. Since they are formulated in the language of differential geometry, they can be referred to as geometric attributes [37], and we can say that they endow the system with an underlying geometric structure. In general, traditional numerical techniques are not concerned with preserving this geometric structure, but are primarily focused on dynamical features such as sensitivity to initial conditions and perturbations (linked to conditioning) and stability (appropriate asymptotic behaviour), as well as computational efficiency. Therefore, in the presence geometric features or invariants, they could lead to simulations displaying non-physical behaviour. For example, they introduce numerical dissipation when used to solve conservative systems, as shown in Figure 1.1.

Geometric numerical integration is a field within numerical analysis that focusses on the development and implementation of geometric or structure-preserving numerical methods. These methods guarantee that the qualitative properties of the continuous-time system are retained in the discrete domain, leading also to improved long-term accuracy in the numerical integration. Drawing upon concepts from geometric mechanics and differential geometry, a

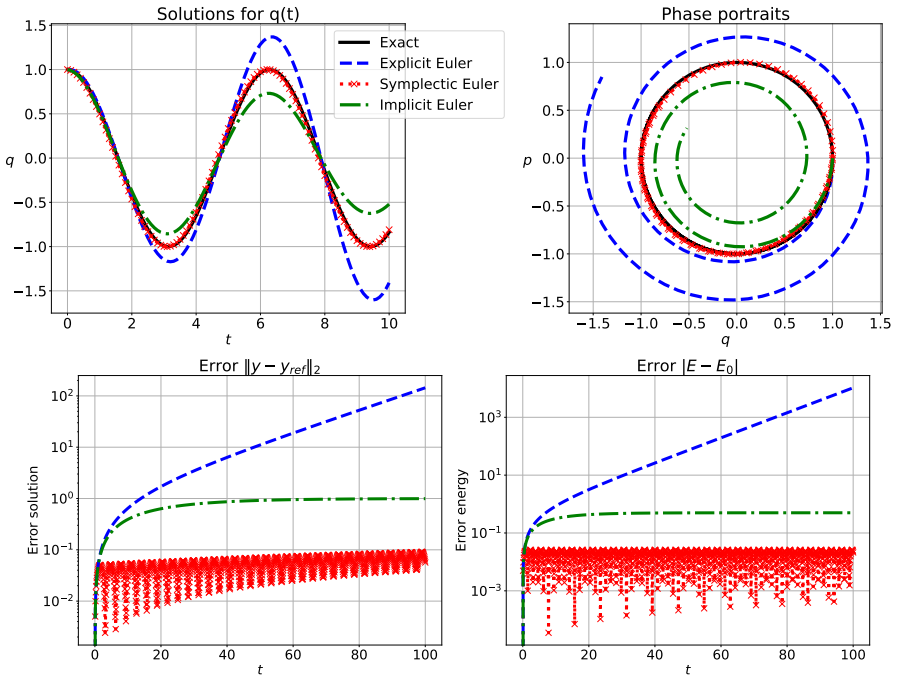


Figure 1.1: Solutions of an ideal mass-spring system $\dot{y} = J\nabla H$, where $y = (q, p)$ and $H = 1/2p^2 + 1/2q^2$, computed using the explicit Euler, implicit Euler, and symplectic Euler methods ([33, sec. I.1.2]). For all the three cases, the initial values are $(q_0, p_0) = (1, 0)$, and the time step size is $h = 0.1$. All three methods are first-order, with the symplectic Euler being a geometric integrator (see section 1.2.1). In the upper right plot we observe that the solutions obtained with the explicit and implicit Euler methods spiral outward and inward, respectively, in the phase space, resulting in an increase or dissipation of energy depicted in the final plot, which shows the error in the energy. In contrast, the solution obtained with the symplectic Euler method results in a closed orbit in the phase space, demonstrating the correct qualitative behaviour. In the bottom plots, we show the long-time behaviour of the solutions in terms of the global error compared to the reference solution, as well as the preservation of energy. Symplectic integrators on Hamiltonian systems show better energy conservation, while other integrators (like the explicit Euler) have errors that grow much more rapidly.

wide range of computational techniques have been developed for solving governing equations of the mechanical systems while respecting their fundamental physics. In this section, the emphasis is on numerical integration techniques that preserve certain geometric properties, like the symplectic form (symplectic

numerical integrators and variational integrators), and the geometry of the manifold of constraints (Lie group integrators). We don't focus on other important properties such as energy or phase-space volume conservation, symmetries, or time-reversibility. We refer to [33] for a comprehensive overview of the topic, and [56] for a recent detailed review.

Let us introduce the following initial value problem

$$\frac{d}{dt}y(t) = f\big(y(t)\big), \quad y(0) = y_0, \tag{1.2.1}$$

where $y : \mathbb{R} \to \mathbb{R}^n$[4]. The map $\varphi_t : \mathbb{R}^n \to \mathbb{R}^n$ that associates the value of the solution at time t with the initial data, i.e.

$$\varphi_t(y_0) = y(t), \text{ with } y(0) = y_0, \tag{1.2.2}$$

is termed the flow map of the system. Let the approximation $y_{n+1} \approx y(t_{n+1})$ be obtained using a numerical one-step method applied to y_n with (constant) stepsize $h = t_{n+1} - t_n$. We call the mapping

$$\Phi_h : y_n \mapsto y_{n+1} \tag{1.2.3}$$

the numerical or discrete flow. For example, $\Phi_h(y_n) = y_n + hf(y_n)$ defines the explicit Euler method.

A function $I(y)$ is called a first integral or constant of motion of the differential equation $\dot{y} = f(y)$, with $\dot{y}$ the time derivative of y and y either a vector or a matrix, if I is constant along solutions $y(t)$ of the differential equations, i.e. $I(y(t)) = I(y_0) =$ constant for all $t > 0$. Differentiating $I(y(t))$ with respect to t gives the condition $I'(y)f(y) = 0$. Geometric numerical integration focusses on numerical methods that respect these geometric invariants or constants of the flow. For example, it is easy to show that every Runge–Kutta method preserves linear invariants of the form $I(y) = d^T y$, with $d \in \mathbb{R}^n$ a constant vector, so that $d^T f(y) = 0 \ \forall y \in \mathbb{R}^n$ (see also [33, p. 99] and references therein). Indeed, let us consider a general s-stages Runge–Kutta scheme

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i K_i, \quad K_i = f\left(y_n + h\sum_{j=1}^{s} a_{i,j}K_j\right), \quad i = 1, \dots, s, \tag{1.2.4}$$

and let us premultiply y_{n+1} by d^T. Since $d^T K_i$ vanish for all i, we have

$$d^T y_{n+1} = d^T y_n + h\sum_{i=1}^{s} b_i d^T K_i = d^T y_n, \tag{1.2.5}$$

[4] We restrict our discussion to autonomous systems. We also assume well posedness of the initial value problem.

that is, the quantity $d^T y$ is conserved by the numerical flow. Furthermore, it can be shown that only a subclass of Runge–Kutta methods preserve all quadratic invariants [33, sec. IV.2], and no Runge–Kutta method conserves all polynomial invariants of degree 3 or higher [33, sec. IV.3].

An example of first integral is provided by the Hamiltonian function (1.1.4), that defines the following canonical Hamiltonian system:

$$\dot{y} = J\nabla H(y), \quad y(0) = y_0, \tag{1.2.6}$$

where $y = (q, p) \in \mathbb{R}^{2n}$ collects positions and conjugate momenta, $q, p \in \mathbb{R}^n$, and

$$J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}, \quad I \text{ the } n \times n \text{ identity matrix.} \tag{1.2.7}$$

Due to the skew-symmetry of the matrix J, we have

$$\dot{H} = \nabla H(y)^T \dot{y} = \nabla H(y)^T J \nabla H(y) = 0. \tag{1.2.8}$$

In other words, the total energy of the system is conserved.

1.2.1 Symplectic integrators

Besides energy conservation, the flow of a Hamiltonian system exhibits another important geometric property, which is the preservation of the following skew-symmetric bilinear form:

$$\omega : \mathbb{R}^{2n} \times \mathbb{R}^{2n} \to \mathbb{R}, \quad \omega(\xi, \eta) = \xi^T J \eta, \tag{1.2.9}$$

that in this case denotes the standard symplectic form of $\mathbb{R}^{2n}$, with J as in (1.2.7). This means that the Jacobian $\Psi := D_y(\varphi_{t,H}(y)) : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ of the flow map $\varphi_{t,H}$ of a canonical Hamiltonian system is a symplectic transformation, that is, it satisfies

$$\Psi^T J \Psi = J \tag{1.2.10}$$

in the domain of definition of Ψ, which implies

$$\omega(\Psi\xi, \Psi\eta) = \omega(\xi, \eta) \text{ for all } \xi, \eta \in \mathbb{R}^{2n}. \tag{1.2.11}$$

In $\mathbb{R}^2$, this means that the flow of the Hamiltonian system is area-preserving: the area of a set of points A in the phase space is the same as the area of $\varphi_{t,H}(A)$. In higher dimensions, the symplecticity of the Hamiltonian flow map implies the conservation of volume in phase space (Liouville's Theorem) [45, p. 55]. Thus, for Hamiltonian systems conserving the symplectic two-form is a stronger property than conserving volume.

Since the symplecticity of the flow is a distinctive feature of Hamiltonian systems [5], it is natural to seek numerical techniques that retain this property. Symplectic methods are specifically referred to as one-step methods $y_{n+1} = \Phi_h(y_n)$ where the numerical flow map Φ_h is symplectic. They result in discrete solutions that exhibit good behaviour, in particular almost preservation of total energy for long time, as already observed in Figure (1.1). This can be explained through the theory of backword error analysis, by which it can be proved that symplectic integrators exactly solve a close modified Hamiltonian system [33, ch. 9]. However, the requirements that need to be met for an integrator to be symplectic are usually strict and result in implicit schemes. Therefore, a key issue is whether symplectic integrators should be favoured over nonsymplectic ones. Symplectic integrators are typically used when the focus is on the qualitative aspects of the solution, while other types of high-order methods may provide sufficiently accurate solutions (see, e.g., [55]).

As a final remark, it is worth noting that a known result in this field indicates that a numerical integrator is unable to preserve both the symplectic structure and the energy at the same time [21, 59]. Thus, for engineering applications it appears to be convenient to classify structure preserving integrators for Hamiltonian systems in two main groups: integrators that preserve energy and integrators that preserve symplecticity [46, 56]. Each method offers its own benefits, and the choice of the most suitable geometric numerical integration technique depends on the specific Hamiltonian system under consideration.

1.2.2 Variational integrators

The idea behind variational integrators is to construct numerical approximations of Lagrangian and Hamiltonian systems by discretising Hamilton's variational principle instead of the governing equations (Euler-Lagrange equations or Hamilton's equations). The resulting numerical scheme is symplectic and momentum-preserving [6] for conservative systems [51, sec. 1.1.2] (see also [46]).

Let us consider a first-order Lagrangian $L : TQ \to \mathbb{R}$ as in (1.1.1) and let us define the action integral

$$\mathcal{S}(q) = \int_{t_0}^{t_N} L\big(q(t), \dot{q}(t)\big)\, dt. \tag{1.2.12}$$

[5] In particular, it can be shown that the flow of a system is symplectic if and only if the system is locally Hamiltonian with respect to a certain Hamiltonian function [33, Theorem 2.6].

[6] Momentum refers to the momentum maps associated with symmetries of the system that, following Noether's theorem, indicate the existence of conserved quantities. Preservation of momentum means that for a discrete system exhibiting symmetry, a discrete version of Noether's theorem identifies a quantity that remains conserved at the discrete level [24, 46].

As mentioned in the previous section, extremising the action over all curves $q(t)$ with fixed endpoints $q(t_0) = q_0$ and $q(t_N) = q_N$ results in the governing Euler-Lagrange equations of the system:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \left(\frac{\partial L}{\partial q}\right) = 0. \tag{1.2.13}$$

In variational integrators, a discrete Lagrangian $L_d : Q \times Q \to \mathbb{R}$ is introduced, with Q the configuration manifold. A trajectory $q : [t_0, t_N] \to Q$ is replaced by a discrete path $q_d : \{t_0, t_0 + h, \ldots, t_0 + Nh = t_N\} \to Q$, where $h \in \mathbb{R}$ is a constant time step, so that $q_n = q_d(t_0 + nh)$ can be considered an approximation to $q(t_0 + nh)$. To discretise the variational principle, a discrete action L_d is defined as

$$L_d(q_n, q_{n+1}) \approx \int_{t_n}^{t_{n+1}} L(q, \dot{q})\, dt, \tag{1.2.14}$$

where the integral over the time interval t_n, t_{n+1} is approximated with a quadrature formula and finite difference schemes are used for the time derivatives. Requiring stationarity of the discrete action sum

$$S_d\left(\{q_n\}_{n=0}^{N}\right) = \sum_{n=0}^{N-1} L_d(q_n, q_{n+1}) \tag{1.2.15}$$

for all variations $\{\delta q_n\}_{n=1}^{N-1}$, with $\{\delta q_0\} = \{\delta q_N\} = 0$, yield the discrete Euler-Lagrange equations

$$D_1 L_d(q_n, q_{n+1}) + D_2 L_d(q_{n-1}, q_n) = 0, \quad n = 1, \ldots, N-1, \tag{1.2.16}$$

that define the time-stepping scheme, with $D_i L_d$ representing the partial differentiation of L_d with respect to the i-th argument.

In the presence of holonomic constraints, the configuration manifold Q is defined as in (1.1.6), with Q embedded in $\mathbb{R}^n$, so that $q \in \mathbb{R}^n$ must satisfy (1.1.5). In this case, the usual approach is to augment the Lagrangian function with Lagrange multipliers $\lambda : [t_0, t_N] \to \mathbb{R}^m$ (see, e.g., [47]) and define

$$\bar{L} : T(\mathbb{R}^n \times \mathbb{R}^m) \to \mathbb{R}, \quad \bar{L}(q, \lambda, \dot{q}, \dot{\lambda}) = L(q, \dot{q}) - g^T(q) \cdot \lambda. \tag{1.2.17}$$

The constrained Euler-Lagrange equations are then obtained as the following set of differential algebraic equations on the ambient space:

$$\frac{\partial L(q, \dot{q})}{\partial (q)} - \frac{d}{dt}\left(\frac{\partial L(q, \dot{q})}{\partial \dot{q}}\right) - G^T(q) \cdot \lambda = 0, \tag{1.2.18}$$

$$g(q) = 0, \tag{1.2.19}$$

where $G(q) = D_q\big(g(q)\big)$ denotes the Jacobian of the constraint function, and the term $-G^T(q)$ represents the constrained forces that act to keep the system on the constraint surface, with their direction and magnitude determined by the Lagrange multipliers $\lambda \in \mathbb{R}^m$. By a similar argument as before, the variational scheme assumes the form of the following constrained discrete Euler-Lagrange equations:

$$\begin{aligned} D_1 L_d\left(q_n, q_{n+1}\right) + D_2 L_d\left(q_{n-1}, q_n\right) - G_d^T\left(q_n\right) \cdot \lambda_n &= 0, \\ g\left(q_{n+1}\right) &= 0, \end{aligned}$$

which are solved for the unknowns $\left(q_{n+1}, \lambda_n\right)$. It should be highlighted that the constraint forces $-G_d^T\left(q_n\right) \cdot \lambda_n$ guarantee the satisfaction of the constraints at the configuration node q_{n+1}.

1.2.3 Numerical methods on manifolds

Let us now consider a differential equation on a $n-m$ dimensional smooth manifold $\mathcal{M}$, that again we consider to be embedded in $\mathbb{R}^n$ and defined implicitly as the zero level set of a function $g : \mathbb{R}^n \to \mathbb{R}^m$ with full rank Jacobian, i.e. $\mathcal{M} = g^{-1}(0)$. Specifically, we consider an initial value problem

$$\dot{y} = f(y), \quad y(0) = y_0 \in \mathcal{M}, \tag{1.2.20}$$

such that $y_0 \in \mathcal{M}$ implies $y(t) \in \mathcal{M}$ for all t, that is, the exact solution lies in $\mathcal{M}$ for all t. This condition is equivalent to (see [33, pag. 115])

$$f(y) \in T_y\mathcal{M} \quad \text{for all } y \in \mathcal{M}. \tag{1.2.21}$$

Therefore, for all $y \in \mathcal{M}$, the vector field $f(y)$ in (1.2.20) defines a mapping from $\mathcal{M}$ to a unique tangent vector in the tangent space $T_y\mathcal{M}$. This situation often arises in practical applications. For example, we observed in the previous section that the Lagrangian vector field of a constrained Lagrangian system is defined on the tangent bundle $TQ \subseteq \mathbb{R}^{2n}$ of the constraint configuration manifold $Q \subseteq \mathbb{R}^n$. The physical solution is then required to satisfy the constraint function, that is, to stay on Q.

Most traditional numerical integration techniques are designed for initial value problems that evolve on a vector space $\mathbb{R}^{2n}$. Hence, an initial strategy to solve the problem would involve directly applying standard numerical integration algorithms to the dynamics in the embedding vector space. Using this approach, however, does not ensure that the numerical solution of an initial-value problem starting on the configuration manifold will remain on it. For example, a Runge–Kutta scheme will evaluate increments in the direction of the vector field (1.2.21), that is tangent to the manifold, hence the approximate solution will deviate from the configuration manifold.

Specific structure-preserving techniques can be employed to force the numerical solution to lie on the manifold (we refer to [33, Chapter 4] for a detailed introduction). A natural approach consists in using a traditional one-step method Φ_h on (1.2.20), with y represented as a vector in the embedding space, and projecting the computed solution $\tilde{y}_{n+1} = \Phi_h(y_n)$ back onto the configuration manifold to obtain y_{n+1}. Usually, this involves solving a constrained minimization problem in some norm $\|\cdot\|$,

$$\min \|y_{n+1} - \tilde{y}_{n+1}\| \quad s.t. \quad g(y_{n+1}) = 0, \tag{1.2.22}$$

with g the constraint function defining the manifold. Therefore, this method can be computationally expensive.

Another significant class of techniques consists in applying the integration scheme on the governing equations expressed in terms of a minimal set of generalised coordinates [33, sec. IV.5]. However, as previously observed, a formulation of the system based on minimal coordinates might be subject to singularities and is in general not globally valid on the manifold. Moreover, in the case of Lagrangian and Hamiltonian systems evolving on a constrained space, it turns out that differential equations expressed in minimal coordinates can be more difficult to derive than those formulated as differential algebraic equations in the embedding space (various examples, like the double pendulum, are provided in [4, 27]).

Finally, we present the underlying idea of geometric numerical methods specifically designed for situations where the manifold is a Lie group. We consider, in particular, the following differential equation

$$\dot{Y} = A(Y)Y, \quad Y(0) = Y_0 \in G \tag{1.2.23}$$

on a matrix Lie group G, with $A : G \to \mathfrak{g}$ a matrix valued function in the Lie algebra $\mathfrak{g}$ of G, which is the tangent space at the identity of the Lie group, $\mathfrak{g} = T_eG$.

Given a differential equation on a Lie group, the key idea of Lie group methods is to write the solution of the differential equation by means of a Lie group action [37]. A comprehensive and well-crafted exposition of these techniques can be found in [36]. Here, we briefly introduce Runge-Kutta-Munthe-Kaas methods (see [52] and references therein), which will be further discussed in paper 1. The basic principle behind these methods consists of expressing the solution of (1.2.23) as

$$Y(t) = \psi(\exp(\sigma(t)), Y_0), \tag{1.2.24}$$

with $\sigma(t)$ defined by a differential equation on the Lie algebra for equations on matrix Lie groups. In (1.2.24), the action ψ reduces to the matrix multiplication

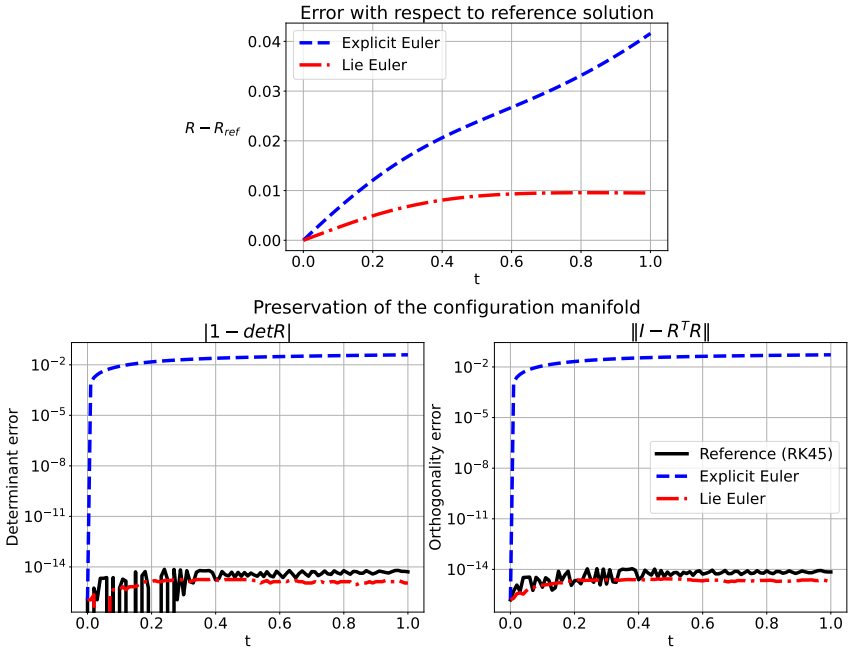


Figure 1.2: Solutions of a differential equation $\dot{R} = A(R)R$ on $SO(3)$ with the explicit Euler method, $R_{n+1} = R_n + hA(R_n)R_n$, and the Lie Euler method, $R_{n+1} = \exp\big(hA(R_n)\big)R_n$, using a time step $h = 0.01$. The reference solution has been computed with the RK45 default method of `scipy.integrate.solve_ivp`, setting relative and absolute tolerances to 10^{-15}. Both the explicit Euler and the Lie Euler methods are of first order. We notice that the solution obtained with the Lie Euler method respects the constraints that define the manifold, i.e. $SO(3) = \{R \in GL(3) \mid R^T R = I,\ detR = 1\}$, to machine accuracy.

$\exp(\sigma(t))Y_0$, i.e., it is intended as the action of the Lie group on itself by left multiplication.

Since the Lie algebra is a linear space, the numerical solution for $\sigma(t)$ can be obtained with standard explicit or implicit Runge-Kutta methods, and we can determine the group element $\exp(\sigma(t))$ that, acting on Y_0, takes the solution to $Y(t)$. This process makes use of the exponential map

$$\exp : \mathfrak{g} \to G, \quad \exp(A) = \sum_{j=0}^{\infty} \frac{A^j}{j}, \tag{1.2.25}$$

which defines a diffeomorphism between a neighbourhood of the origin in $\mathfrak{g}$ and a neighbourhood of the identity in G. This actually means that the exponential map provides a local parameterisation near the identity of the Lie group, allowing us to identify a point on G with a set of coordinates on $\mathfrak{g}$, called in this

case the canonical coordinates of the first kind. Other examples of parameterisation are possible, such as the Cayley transform and the canonical coordinates of the second kind [33, Section IV.8.3].

It is worth mentioning that a matrix Lie algebra comes with the bilinear antisymmetric map

$$[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}, \quad [A, B] = AB - BA. \tag{1.2.26}$$

known as the Lie bracket or commutator, and that σ satisfies an initial value problem where the vector field is given in terms of nested commutators, as in (2.2.6) in paper 1, and $\sigma(0) = 0 \in \mathfrak{g}$. Since the Lie algebra is closed under matrix addition, scalar multiplication, and the commutator operation defined above, we have that all operations involved in a Runge–Kutta method with step size h (like evaluation of the vector field and computation of updates) produce results in the Lie algebra, in particular the one-step approximation $\tilde{\sigma} \approx \sigma(h)$ which is used to obtain the next approximation via the action $\exp(\tilde{\sigma}) Y_0 \in G$. In figure 1.2 we show an example of differential equation on $SO(3)$ solved with the explicit Euler and the Lie Euler methods. The latter is a Lie group integrator based on the standard explicit Euler scheme to get an approximation of $\sigma(h)$ on the Lie algebra.

We also remark that, under some assumptions on the approximation of $\sigma(t)$, the order of the underlying classical Runge-Kutta method is preserved (see, e.g., [36, pag. 42] or [33, Theorem 8.5]). This can be considered as an outcome of interpreting (1.2.24) as a smooth local change of variables that transform (1.2.23) into an initial value problem on a linear space.

This method will be explained for the more general case of vector fields on homogeneous manifolds in paper 1 . The setting will remain mostly the same, except that a function $f : \mathcal{M} \to \mathfrak{g}$ is now being introduced, which reduces to $A(Y)$ in the case of equation (1.2.23). Most of the codes are available in THREAD's GitHub repository [2].

1.3 Data-driven methods in engineering

In recent years, advances in machine learning technologies, especially deep neural networks, have been significant across various scientific and engineering domains. This progress has been made possible, among others, by the abundance of data from physical systems measurements, the decreasing cost of sensors, the increase in computational capabilities, and the improvement of data storage technologies. As a result, data-driven methodologies are prompting a shift in the way many problems in science and engineering are approached (we refer to [11] for a thorough introduction on the subject). Notably, they have

proven to be effective in modelling mechanical systems and in making predictions about their states, as we investigate in paper 4 with respect to the static equilibria of Euler's elastica.

Machine learning methods are useful for identifying underlying relationships and hidden patterns within a dataset. On one side, this approach can help to improve existing models, for example identifying unknown model parameters, or to discover governing equations of dynamical systems [12]. On the other side, methods that extract informative patterns, for example uncovering correlations between different variables, can provide guidance for decision-making in various engineering fields, as we study in paper 5.

Usually, data are gathered in multidimensional arrays that exhibit low-rank ([11, p. 3]). Therefore, a key goal is to identify the underlying low-dimensional feature space in which the data can be represented (this assumption is also called the manifold hypothesis, see e.g. [30, sec. 3.6.2] or [28, sec. 5.11.3]). This procedure can be completely data-driven or can be enhanced by feature engineering guided by expert knowledge. By features, we mean all the attributes of measurements or observations collected into data points. Feature engineering involves the process of converting original features into new ones that are more relevant to the specific task, i.e., that can be useful to develop better classifiers or predictors. A few examples are shown in paper 5, where principal component analysis and correlation are crucial in the preprocessing of data.

An important aspect to consider is that the majority of machine learning methods are framed as optimisation problems, with the aim of maximising the accuracy of regression and classification models [11]. In deep learning, the model is formulated as a neural network f_θ, which is a composition of parametric maps applied to the input x. The parameters θ are determined by solving an optimisation problem

$$\operatorname*{argmin}_\theta \mathcal{L}\big(f_\theta(x)\big). \tag{1.3.1}$$

Here, $\mathcal{L}$ represents a cost function that must be minimised, usually called loss. The solution of the optimisation problem via iterative methods is called training. The parameters are determined by evaluating how well the model fits the data during the training process, and subsequently the optimal model is selected based on its capacity to generalise to new, unseen data. Various network architectures (i.e., how the compositional maps are defined) will be presented in papers 3, 4, and 5. Here, we provide a short overview of the minimisation procedure, and refer to [11, 34, 40, 42] for further details.

A cost function usually involves summing individual terms $\mathcal{C}_{x^i}$, $i = 1, \dots, N$, across the training data x^i and taking the mean, for example

$$\mathcal{L}(\theta) = \frac{1}{M} \sum_{i=1}^{N} \mathcal{C}_{x^i}(\theta), \tag{1.3.2}$$

as in (5.4.1) in paper 4. In (1.3.2) we explicitly write the loss as a function of the parameters θ. Minimisation is typically performed using gradient descent, which requires calculating the gradients of the loss function with respect to all the parameters in the neural network. The learning process typically employs stochastic gradient descent methods (see [30, p. 15] and references therein), where the idea is to substitute the real gradient with an approximation obtained from a randomly chosen subset of the data, called batch. Therefore, we choose s integers $\{k_1, k_2, \ldots, k_s\}$, $s \ll N$, uniformly at random from $\{1, 2, \ldots, N\}$, and we define the update rule as

$$\theta \mapsto \theta - \eta \frac{1}{s} \sum_{i=1}^{s} \nabla \mathcal{C}_{x^{k_i}}(\theta), \tag{1.3.3}$$

where the learning rate η determines the size of the step in the direction of the negative gradient and is crucial in determining how fast the algorithm converges to a minimum. In (1.3.3), the set $\left\{x^{k_i}\right\}_{i=1}^{s}$ is the batch, with s the batch size. The batch aims to provide a statistically meaningful sample of the complete training dataset and changes with each iteration. It is possible for the same sample to be selected multiple times in subsequent steps for different batches, or alternatively, the training set can be randomly split into distinct batches. In either case, the algorithm cycles through the batches until all training inputs are used, corresponding to an epoch of training. At that point, it starts over with a new training epoch.

The number of training epochs, as well as the batch size and the learning rate, are known as hyperparameters, in order to distinguish them from the parameters of the neural network. Their selection, along with additional architectural choices, is made with the aim of speeding up convergence and improving expressiveness, and could also be adjusted dynamically during the training process.

The volume of parameters and training data tends to be extremely large in deep learning, making it costly to calculate the gradient vector of the loss function during each iteration of the gradient descent method. Irrespective of the optimisation method used (such as stochastic gradient descent or one of its adaptations like the widely used Adam algorithm [39]), neural networks make use of an efficient algorithm called backpropagation (see [30, p. 14] and references therein) to compute gradients of the loss function. Backpropagation is an automatic differentiation technique and, in particular, can be considered an instance of reverse-mode differentiation. It is based on the chain rule and leverages the hierarchical structure of neural networks to calculate gradients (a basic introduction is presented in [34] and [40]).

In the experiments of this thesis, we use the deep learning framework provided by `PyTorch` to define and train deep learning models. A key feature of

`PyTorch` is that it records all computations in a directed acyclic graph and enables reverse-mode automatic differentiation, making it ideal for efficiently and accurately computing derivatives in functions that exhibit graph-like structures, such as neural networks.

It should be remarked that, in the context of deep learning, we usually deal with challenging non-convex optimization problems [30, p. 57] (see also [48]), whose solutions depend on the specific definition of the loss function and network architecture, and on the appropriate choice of the optimisation algorithm. Striking the right balance between training ease and network expressiveness, i.e. the class of functions that can be approximated by the network, is crucial. Regarding the second aspect, theoretical results, known as universal approximation theorems, demonstrate that neural networks possess significant expressive capabilities. For example, they show that neural networks have the ability to approximate any continuous multi-input multi-output function with arbitrary precision (see, e.g., [40, Sec. 3.1], [9, Sec. 2.1], and references therein). However, their practical relevance is limited, as they do not offer assurance such neural network and the appropriate parameters can be found. Finally, in order for the neural network to generalise well, a range of techniques known as regularization strategies can be utilized. These usually involve extra terms in the definition of the loss and can be helpful for creating informative representations and avoiding overfitting, which occurs when a model performs well on the training data, but poorly on new, unseen inputs. A few examples are provided in paper 4, e.g. in the definition of the loss (5.5.1). A recent trend involves enhancing data-driven models with physical principles and geometric properties, such as conservation laws, constraints and symmetries. This results in hybrid approaches that improve the reliability of the data-driven models and alleviate their lack of interpretability, especially in the case of deep neural networks. An example of encoding the physics in both the definition of the network architecture and the learning procedure is shown in paper 3.

1.4 Summary of papers

Paper 1: Lie group integrators for mechanical systems

Elena Celledoni, Ergys Çokaj, Andrea Leone,
Davide Murari, Brynjulf Owren

International Journal of Computer Mathematics 99(1), 58–88

In this article, we provide a review of Lie group integration methods for mechanical systems, whose dynamics can be described by a Lie group acting on a manifold. We set up the necessary differential geometric background, with

a specific emphasis on the Lie groups $SO(3)$ and $SE(3)$, and perform various numerical experiments using two classes of numerical schemes, RKMK and commutator-free Lie group integrators. In particular, we consider different Lie group formulations of the heavy top equations, a chain of pendulums, and a multibody system model for drone dynamics. In the last two cases, we show that the configuration manifold is homogeneous and write the corresponding transitive action on the velocity phase space. The numerical experiments show how Lie group integrators preserve the geometry of the entire phase space to machine accuracy, much better than classical methods like ODE45 or the Runge–Kutta 4.

Paper 2: Dynamics of the N-fold Pendulum in the Framework of Lie Group Integrators

Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, Brynjulf Owren

Progress in Industrial Mathematics at ECMI 2021. Mathematics in Industry, vol 39. Springer Cham, pp. 297-304

This short conference proceedings paper is an extension of the previous one. After an overview of RKMK methods, we better clarify how to extend Lie group actions to Cartesian products of homogeneous manifolds. We then focus on variable step-size methods, showing how to implement them in a Lie group context. We perform numerical experiments considering the example of the chain of pendulums, and we compare constant- and adaptive-step-size RKMK methods.

Paper 3: Learning Hamiltonians of constrained mechanical systems

Elena Celledoni, Andrea Leone, Davide Murari, Brynjulf Owren

Journal of Computational and Applied Mathematics 417, 114608

This article combines the structure-preserving numerical methods introduced before with a data-driven approach for modelling Hamiltonian mechanical systems. In particular, we describe a neural network-based approach to approximate (learn) the Hamiltonian of mechanical systems with holonomic constraints, given trajectory data. In the learning process, we integrate the resulting Hamilton's equations to match the given trajectories. We investigate whether this learning strategy profits from integration methods that preserve the constraints, specifically from RKMK methods.

Paper 4: Neural networks for the approximation of Euler's elastica

Elena Celledoni, Ergys Çokaj, Andrea Leone,
Sigrid Leyendecker, Davide Murari, Brynjulf Owren,
Rodrigo T. Sato Martín de Almagro, Martina Stavole

Submitted

In this paper, we address the problem of approximating solutions of the planar Euler's elastica in the static regime with neural networks. We propose two approaches in which we learn discrete and continuous approximations of the solutions, discussing limitations and advantages in both cases. We present numerical experiments showing that the proposed neural networks can effectively approximate configurations of the planar Euler's elastica for a range of different boundary conditions and have good generalisation capability. Determining the static equilibria of an inextensible planar Euler elastica, described by a constrained second order Lagrangian, is a classical problem of beam theory. This work showcases both the potential and limitations of deep learning in industrial settings involving flexible systems that experience large bending deformations.

Paper 5: Supervised time series classification for anomaly classification in subsea engineering

Ergys Çokaj, Halvor Snersrud Gustad, Andrea Leone,
Per Thomas Moe, Lasse Moldestad

To appear on the Journal of Computational Dynamics

This article compares different data-based approaches to perform binary classification of multi-sensor time series signals. The goal is to detect a systematic change in the structural response of a subsea production system exposed to operational loads. More specifically, given a synthetic data set based on sensor simulations, we determine the status (broken versus intact) of subsea oil wells. We start by explaining how to preprocess the data for effective feature extraction. Then, we apply both traditional statistical methods and modern machine learning techniques, discussing the advantages and drawbacks of each. Despite using established algorithms, it is important to note that we apply them to a specific type of time series data, mimicking real sensor measurements from physical systems to ensure practical relevance. The significance of this study in the offshore oil industry lies in the importance of preventing any loss of structural integrity and pressure control caused by wellhead cracking. We show that machine learning algorithms can provide advantages by reducing the need for human decision-making.

Statement on the layout of the thesis and the use of AI tools. The layout, bibliography, and presentation style of the papers have been unified, and some typographical errors have been rectified. To conform to the B5 format of this thesis, some equations have been reformatted. No other substantial modifications have been made.

The thesis has been written using the LaTeX editor Overleaf, with the help of the Writefull extension for language feedback and grammar check.

Bibliography

[1] Joint training on numerical modelling of highly flexible structures for industrial applications. URL: https://cordis.europa.eu/project/id/860124. 1

[2] Thread deliverable 3.2. URL: https://github.com/THREAD-3-2. 16

[3] Stuart S. Antman. *Nonlinear Problems of Elasticity*. Applied Mathematical Sciences. Springer New York, 2005. 1

[4] Martin Arnold. *DAE Aspects of Multibody System Dynamics*, pages 41–106. 03 2017. doi:10.1007/978-3-319-46618-7_2. 1, 5, 14

[5] Vladimir I. Arnold. *Mathematical methods of classical mechanics*, volume 60 of *Graduate Texts in Mathematics*. Springer New York, 1989. 3

[6] Olivier A. Bauchau and Shilei Han. Interpolation of rotation and motion. *Multibody System Dynamics*, 31:339–370, 2014. 1

[7] Martin Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems. *IFAC-PapersOnLine*, 54(9):620–623, 2021. 2

[8] Sindre Stenen Blakseth, Adil Rasheed, Trond Kvamsdal, and Omer San. Combining physics-based and data-driven techniques for reliable hybrid analysis and modeling using the corrective source term approach. *Applied Soft Computing*, 128:109533, 2022. 2

[9] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. arXiv:2104.13478. 19, 167

[10] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 2

[11] Steven L. Brunton and J. Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022. 2, 16, 17

[12] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016. 17, 92

[13] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Dynamics of the N-fold pendulum in the framework of Lie group integrators. In *Progress in Industrial Mathematics at ECMI 2021*, pages 297–304, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-11818-0_39. 2

[14] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Lie group integrators for mechanical systems. *International Journal of Computer Mathematics*, 99(1):58–88, 2022. doi:10.1080/00207160.2021.1966772. 2

[15] Elena Celledoni, Matthias J. Ehrhardt, Christian Etmann, Robert I. McLachlan, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Structure-preserving deep learning. *European journal of applied mathematics*, 32(5):888–936, 2021. 2

[16] Elena Celledoni, Matthias J. Ehrhardt, Christian Etmann, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Equivariant neural networks for inverse problems. *Inverse Problems*, 37(8):085006, 2021. 2

[17] Elena Celledoni, Andrea Leone, Davide Murari, and Brynjulf Owren. Learning Hamiltonians of constrained mechanical systems. *Journal of Computational and Applied Mathematics*, 417:114608, 2023. doi:10.1016/j.cam.2022.114608. 3

[18] Elena Celledoni, Håkon Marthinsen, and Brynjulf Owren. An introduction to Lie group integrators–basics, new developments and applications. *Journal of Computational Physics*, 257:1040–1061, 2014. 2, 97

[19] Elena Celledoni, Davide Murari, Brynjulf Owren, Carola-Bibiane Schönlieb, and Ferdia Sherry. Dynamical systems–based neural networks. *SIAM Journal on Scientific Computing*, 45(6):A3071–A3094, 2023. 2

[20] Elena Celledoni, Ergys Çokaj, Andrea Leone, Sigrid Leyendecker, Davide Murari, Brynjulf Owren, Rodrigo T. Sato Martín de Almagro, and Martina Stavole. Neural networks for the approximation of Euler's elastica, 2023. arXiv:2312.00644. 3

[21] Philippe Chartier, Erwan Faou, and Ander Murua. An algebraic approach to invariant preserving integators: the case of quadratic and Hamiltonian invariants. *Numerische Mathematik*, 103:575–590, 2006. 11

[22] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016. 2

[23] Ergys Çokaj, Halvor Snersrud Gustad, Andrea Leone, Per Thomas Moe, and Lasse Moldestad. Supervised time series classification for anomaly detection in subsea engineering, 2024. arXiv:2403.08013. 3

[24] François Demoures, François Gay-Balmaz, Sigrid Leyendecker, Sina Ober-Blöbaum, Tudor S. Ratiu, and Yves Weinand. Discrete variational Lie group formulation of geometrically exact beam dynamics. *Numerische Mathematik*, 130:73–123, 2015. 11

[25] Eva Dierkes, Christian Offen, Sina Ober-Blöbaum, and Kathrin Flaßkamp. Hamiltonian neural networks with automatic symmetry detection. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(6), 2023. 2

[26] Sølve Eidnes, Alexander J. Stasik, Camilla Sterud, Eivind Bøhn, and Signe Riemer-Sørensen. Pseudo-Hamiltonian neural networks with state-dependent external forces. *Physica D: Nonlinear Phenomena*, 446:133673, 2023. 2

[27] Marc Finzi, Ke Alexander Wang, and Andrew G. Wilson. Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. *Advances in neural information processing systems*, 33:13880–13889, 2020. 14, 85, 89, 96, 100

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 17

[29] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019. 2, 85, 86, 96

[30] Philipp Grohs and Gitta Kutyniok, editors. *Mathematical Aspects of Deep Learning*. Cambridge University Press, 2022. 17, 18, 19

[31] Yiqi Gu and Michael K. Ng. Deep neural networks for solving large linear systems arising from high-dimensional problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023. 2, 111, 112

[32] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017. 2

[33] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2006. 2, 3, 5, 8, 9, 10, 11, 13, 14, 16

[34] Catherine F. Higham and Desmond J. Higham. Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891, 2019. 17, 18, 117

[35] Darryl D. Holm, Tanya Schmah, and Cristina Stoica. *Geometric mechanics and symmetry: from finite to infinite dimensions*, volume 12. Oxford University Press, 2009. 3, 4, 5, 6

[36] Arieh Iserles, Hans Z. Munthe-Kaas, Syvert P. Nørsett, and Antonella Zanna. Lie-group methods. *Acta numerica*, 9:215–365, 2000. 5, 14, 16, 97, 98

[37] Arieh Iserles and G. Reinout W. Quispel. Why geometric numerical integration? In *Discrete Mechanics, Geometric Integration and Lie–Butcher Series: DMGILBS, Madrid, May 2015*, pages 1–28. Springer, 2018. 7, 14

[38] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. 2

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 18

[40] Stefan Kollmannsberger, Davide D'Angella, Moritz Jokeit, and Leon Herrmann. *Deep learning in computational mechanics. An introductory course*. Springer, 2021. 17, 18, 19, 111, 112, 117

[41] Holger Lang, Joachim Linn, and Martin Arnold. Multi-body dynamics simulation of geometrically exact Cosserat rods. *Multibody System Dynamics*, 25(3):285–312, 2011. 1, 7

[42] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015. 17

[43] John M. Lee. *Introduction to smooth manifolds*. Springer, 2012. 3, 86, 89

[44] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Global formulations of Lagrangian and Hamiltonian dynamics on manifolds. *Springer*, 13:31, 2017. 1, 3, 4, 5, 6, 85, 88, 99

[45] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Number 14. Cambridge university press, 2004. 10

[46] Adrian Lew, Jerrold E. Marsden, Michael Ortiz, and Matthew West. An overview of variational integrators, August 2023. 11

[47] Sigrid Leyendecker, Jerrold E. Marsden, and Michael Ortiz. Variational integrators for constrained dynamical systems. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: Applied Mathematics and Mechanics*, 88(9):677–708, 2008. 12

[48] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018. 19

[49] Yana Lishkova, Paul Scherer, Steffen Ridderbusch, Mateja Jamnik, Pietro Liò, Sina Ober-Blöbaum, and Christian Offen. Discrete Lagrangian neural networks with automatic symmetry discovery. *IFAC-PapersOnLine*, 56(2):3203–3210, 2023. 2

[50] Jerrold E. Marsden and Tudor S. Ratiu. *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*. Texts in Applied Mathematics. Springer New York, 2013. 3, 4

[51] Jerrold E. Marsden and Matthew West. Discrete mechanics and variational integrators. *Acta numerica*, 10:357–514, 2001. 11

[52] Hans Munthe-Kaas. High order runge-kutta methods on manifolds. *Applied Numerical Mathematics*, 29(1):115–127, 1999. 14

[53] Vien Minh Nguyen-Thanh, Xiaoying Zhuang, and Timon Rabczuk. A deep energy method for finite deformation hyperelasticity. *European Journal of Mechanics-A/Solids*, 80:103874, 2020. 2

[54] Haakon Robinson, Suraj Pawar, Adil Rasheed, and Omer San. Physics guided neural networks for modelling of non-linear dynamics. *Neural Networks*, 154:333–345, 2022. 2

[55] Jesus M. Sanz-Serna and Mari P. Calvo. *Numerical Hamiltonian Problems*. Applied mathematics and mathematical computation. Dover Publications, 2018. 11

[56] Harsh Sharma, Mayuresh Patil, and Craig Woolsey. A review of structure-preserving numerical methods for engineering applications. *Computer Methods in Applied Mechanics and Engineering*, 366:113067, 2020. 9, 11

[57] Valentin Sonneville, Alberto Cardona, and Olivier Brüls. Geometrically exact beam finite element formulated on the special Euclidean group SE(3). *Computer Methods in Applied Mechanics and Engineering*, 268:451–474, 2014. 1

[58] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11, 2017. 2

[59] Ge Zhong and Jerrold E. Marsden. Lie-Poisson Hamilton-Jacobi theory and Lie-Poisson integrators. *Physics Letters A*, 133(3):134–139, 1988. 11

Lie group integrators for mechanical systems

Elena Celledoni, Ergys Çokaj, Andrea Leone,
Davide Murari and Brynjulf Owren

International Journal of Computer Mathematics, 2022, 99.1: 58-88

Lie Group integrators for mechanical systems

Abstract. Since they were introduced in the 1990s, Lie group integrators have become a method of choice in many application areas. These include multibody dynamics, shape analysis, data science, image registration and biophysical simulations. Two important classes of intrinsic Lie group integrators are the Runge–Kutta–Munthe–Kaas methods and the commutator free Lie group integrators. We give a short introduction to these classes of methods. The Hamiltonian framework is attractive for many mechanical problems, and in particular we shall consider Lie group integrators for problems on cotangent bundles of Lie groups where a number of different formulations are possible. There is a natural symplectic structure on such manifolds and through variational principles one may derive symplectic Lie group integrators. We also consider the practical aspects of the implementation of Lie group integrators, such as adaptive time stepping. The theory is illustrated by applying the methods to two nontrivial applications in mechanics. One is the N-fold spherical pendulum where we introduce the restriction of the adjoint action of the group $SE(3)$ to TS^2, the tangent bundle of the two-dimensional sphere. Finally, we show how Lie group integrators can be applied to model the controlled path of a payload being transported by two rotors. This problem is modeled on $\mathbb{R}^6 \times \big(SO(3) \times \mathfrak{so}(3)\big)^2 \times (TS^2)^2$ and put in a format where Lie group integrators can be applied.

2.1 Introduction

In many physical problems, including multi-body dynamics, the configuration space is not a linear space, but rather consists of a collection of rotations and translations. A simple example is the free rigid body whose configuration space consists of rotations in 3D. A more advanced example is the simplified model of the human body, where the skeleton at a given time is described as a system of interacting rods and joints. Mathematically, the structure of such problems is usually best described as a manifold. Since manifolds by definition can be equipped with local coordinates, one can always describe and simulate such systems locally as if they were linear spaces. There are of course many choices of local coordinates, for rotations some famous ones are: Euler angles, the Tait-Bryan angles commonly used in aerospace applications, the unit length quaternions, and the exponentiated skew-symmetric 3×3-matrices. Lie group integrators represent a somewhat different strategy. Rather than specifying a choice of local coordinates from the outset, in this approach the model and the numerical integrator are expressed entirely in terms of a Lie group and its action on the phase space. This often leads to a more abstract and simpler formulation of the mechanical system and of the numerical schemes, deferring further details to the implementation phase.

In the literature one can find many different types and formats of Lie group integrators. Some of these are completely general and intrinsic, meaning that they only make use of inherent properties of Lie groups and manifolds as was suggested in [6, 11, 40]. But many numerical methods have been suggested that add structure or utilise properties which are specific to a particular Lie group or manifold. Notable examples of this are the methods based on canonical coordinates of the second kind [45], and the methods based on the Cayley transformation [13, 31], applicable e.g. to the rotation groups and Euclidean groups. In some applications e.g. in multi-body systems, it may be useful to formulate the problem as a mix between Lie groups and kinematic constraints, introducing for instance Lagrange multipliers. Sometimes this may lead to more practical implementations where a basic general setup involving Lie groups can be further equipped with different choices of constraints depending on the particular application. Such constrained formulations are outside the scope of the present paper. It should also be noted that the Lie group integrators devised here do not make any a priori assumptions about how the manifold is represented.

The applications of Lie group integrators for mechanical problems also have a long history, two of the early important contributions were the Newmark methods of Simo and Vu–Quoc [49] and the symplectic and energy-momentum methods by Lewis and Simo [31]. Mechanical systems are often described as Euler–Lagrange equations or as Hamiltonian systems on manifolds, with or without external forces, [28]. Important ideas for the discretization of mechanical systems originated also from the work of Moser and Veselov [37, 51] on discrete integrable systems. This work served as motivation for further developments in the field of geometric mechanics and for the theory of (Lie group) discrete variational integrators [20, 27, 29]. The majority of Lie group methods found in the literature are one-step type generalisations for classical methods, such as Runge–Kutta type formulas. In mechanical engineering, the classical BDF methods have played an important role, and were recently generalised [54] to Lie groups. Similarly, the celebrated α-method for linear spaces proposed by Hilber, Hughes and Taylor [22] has been popular for solving problems in multibody dynamics, and in [1, 2, 4] this method is generalised to a Lie group integrator.

The literature on Lie group integrators is rich and diverse, the interested reader may consult the surveys [7, 10, 26, 44] and Chapter 4 of the monograph [18] for further details.

In this paper we discuss different ways of applying Lie group integrators to simulating the dynamics of mechanical multi-body systems. Our point of departure is the formulation of the models as differential equations on manifolds. Assuming to be given either a Lie group acting transitively on the manifold

$\mathcal{M}$ or a set of frame vector fields on $\mathcal{M}$, we use them to describe the mechanical system and further to build the numerical integrator. We shall here mostly consider schemes of the types commonly known as Crouch–Grossman methods [11], Runge–Kutta–Munthe–Kaas methods [39, 40] and Commutator-free Lie group methods [6].

The choice of Lie group action is often not unique and thus the same mechanical system can be described in different equivalent ways. Under numerical discretization the different formulations can lead to the conservation of different geometric properties of the mechanical system. In particular, we explore the effect of these different formulations on a selection of examples in multibody dynamics. Lie group integrators have been succesfully applied for the simulation of mechanical systems, and in problems of control, bio-mechanics and other engineering applications, see for example [46], [27] [9], [25]. The present work is motivated by applications in modeling and simulation of slender structures like Cosserat rods and beams [49], and one of the examples presented here is the application to a chain of pendula. Another example considers an application for the controlled dynamics of a multibody system.

In section 2.2 we give a review of the methods using only the essential intrinsic tools of Lie group integrators. The algorithms are simple and amenable for a coordinate-free description suited to object oriented implementations. In section 2.3, we discuss Hamiltonian systems on Lie groups, and we present three different Lie group formulations of the heavy top equations. These systems (and their Lagrangian counterpart) often arise in applications as building blocks of more realistic systems which comprise also damping and control forces. In section 2.4, we discuss some ways of adapting the integration step size in time. In section 2.5 we consider the application to a chain of pendula. And in section 2.6 we consider the application of a multi-body system of interest in the simulation and control of drone dynamics.

2.2 Lie group integrators

2.2.1 The formulation of differential equations on manifolds

Lie group integrators solve differential equations whose solution evolve on a manifold $\mathcal{M}$. For ease of notation we restrict the discussion to the case of autonomous vector fields, although allowing for explicit t-dependence could easily have been included. This means that we seek a curve $y(t) \in \mathcal{M}$ whose tangent at any point coincides with a vector field $F \in \mathcal{X}(\mathcal{M})$ and passing through a designated initial value y_0 at $t = t_0$

$$\dot{y}(t) = F|_{y(t)}, \qquad y(t_0) = y_0. \tag{2.2.1}$$

Before addressing numerical methods for solving (2.2.1) it is necessary to introduce a convenient way of representing the vector field F. There are different ways of doing this. One is to furnish $\mathcal{M}$ with a transitive action $\psi : G \times \mathcal{M} \to \mathcal{M}$ by some Lie group G of dimension $d \geq \dim \mathcal{M}$. We denote the action of g on m as $g \cdot m$, i.e. $g \cdot m = \psi(g, m)$. Let $\mathfrak{g}$ be the Lie algebra of G, and denote by $\exp : \mathfrak{g} \to G$ the exponential map. We define $\psi_* : \mathfrak{g} \to \mathcal{X}(\mathcal{M})$ to be the infinitesimal generator of the action, i.e.

$$F_\xi\Big|_m = \psi_*(\xi)\big|_m = \left.\frac{d}{dt}\right|_{t=0} \psi(\exp(t\xi), m). \tag{2.2.2}$$

The transitivity of the action now ensures that $\psi_*(\mathfrak{g})\big|_m = T_m\mathcal{M}$ for any $m \in \mathcal{M}$, such that any tangent vector $v_m \in T_m\mathcal{M}$ can be represented as $v_m = \psi_*(\xi_v)\big|_m$ for some $\xi_v \in \mathfrak{g}$ (ξ_v may not be unique). Consequently, for any vector field $F \in \mathcal{X}(\mathcal{M})$ there exists a map $f : \mathcal{M} \to \mathfrak{g}$[1] such that

$$F|_m = \psi_*(f(m))\big|_m, \quad \text{for all } m \in \mathcal{M}. \tag{2.2.3}$$

This is the original tool [40] for representing a vector field on a manifold with a group action. Another approach was used in [11] where a set of *frame vector fields* $E_1, \ldots, E_d$ in $\mathcal{X}(\mathcal{M})$ was introduced assuming that for every $m \in \mathcal{M}$,

$$\operatorname{span}\{E_1\big|_m, \ldots, E_d\big|_m\} = T_m\mathcal{M}.$$

Then, for any vector field $F \in \mathcal{X}(\mathcal{M})$ there are, in general non-unique, functions $f_i : \mathcal{M} \to \mathbb{R}$, which can be chosen with the same regularity as F, such that

$$F|_m = \sum_{i=1}^{d} f_i(m)\, E_i\big|_m.$$

A fixed vector $\xi \in \mathbb{R}^d$ will define a vector field F_ξ on $\mathcal{M}$ similar to (2.2.2)

$$F_\xi\Big|_m = \sum_{i=1}^{d} \xi_i E_i|_m. \tag{2.2.4}$$

If $\xi_i = f_i(p)$ for some $p \in \mathcal{M}$, the corresponding F_ξ will be a vector field in the linear span of the frame which coincides with F at the point p. Such a vector field was named by [11] as a *the vector field frozen at* p.

The two formulations just presented are in many cases connected, and can then be used in an equivalent manner. Suppose that $e_1, \ldots, e_d$ is a basis of the

[1] If the Lie group action is smooth, a map f of the same regularity as F can be found [53]

Lie algebra $\mathfrak{g}$, then we can simply define frame vector fields as $E_i = \psi_*(e_i)$ and the vector field we aim to describe is,

$$F|_m = \psi_*(f(m))\big|_m = \psi_*(\sum_i f_i(m)e_i)\Big|_m = \sum_i f_i\ E_i\big|_m.$$

As mentioned above there is a non-uniqueness issue when defining a vector field by means of a group action or a frame. A more fundamental description can be obtained using the machinery of connections. The assumption is that the simply connected manifold $\mathcal{M}$ is equipped with a connection which is flat and has constant torsion. Then F_p, the frozen vector field of F at p defined above, can be defined as the unique element $F_p \in \mathcal{X}(\mathcal{M})$ satisfying

1. $F_p|_p = F|_p$,
2. $\nabla_X F_p = 0$ for any $X \in \mathcal{X}(M)$.

So F_p is the vector field that coincides with F at p and is parallel transported to any other point on $\mathcal{M}$ by the connection ∇. Since the connection is flat, the parallel transport from the point p to another point $m \in \mathcal{M}$ does not depend on the chosen path between the two points. For further details, see e.g. [32].

Example 1. *For mechanical systems on Lie groups, two important constructions are the adjoint and coadjoint representatons. For every $g \in G$ there is an automorphism $\mathrm{Ad}_g : \mathfrak{g} \to \mathfrak{g}$ defined as*

$$\mathrm{Ad}_g(\xi) = TL_g \circ TR_{g^{-1}}(\xi),$$

where L_g and R_g are the left and right multiplications respectively, $L_g(h) = gh$ and $R_g(h) = hg$. Since Ad *is a representation, i.e.* $\mathrm{Ad}_{gh} = \mathrm{Ad}_g \circ \mathrm{Ad}_h$ *it also defines a left Lie group action by G on $\mathfrak{g}$. From this definition and a duality pairing $\langle\cdot,\cdot\rangle$ between $\mathfrak{g}$ and $\mathfrak{g}^*$, we can also derive a representation on $\mathfrak{g}^*$ denoted Ad^*_g, simply by*

$$\langle \mathrm{Ad}^*_g(\mu), \xi\rangle = \langle \mu, \mathrm{Ad}_g(\xi)\rangle, \quad \xi \in \mathfrak{g},\ \mu \in \mathfrak{g}^*.$$

The action $g \cdot \mu = \mathrm{Ad}^*_{g^{-1}}(\mu)$ has infinitesimal generator given as

$$\psi_*(\xi)\big|_\mu = -\mathrm{ad}^*_\xi \mu.$$

Following [34], for a Hamiltonian $H : T^*G \to \mathbb{R}$, define H^- to be its restriction to $\mathfrak{g}^*$. Then the Lie-Poisson reduction of the dynamical system is defined on $\mathfrak{g}^*$ as

$$\dot{\mu} = -\mathrm{ad}^*_{\frac{\partial H^-}{\partial \mu}} \mu,$$

and this vector field is precisely of the form (2.2.3) *with* $f(\mu) = \frac{\partial H^-}{\partial \mu}(\mu)$. *A side effect of this is that the integral curves of these Lie-Poisson systems preserve coadjoint orbits, making the coadjoint action an attractive choice for Lie group integrators.*

Let us now detail the situation for the very simple case where $G = SO(3)$. *The Lie algebra* $\mathfrak{so}(3)$ *can be modeled as* 3×3 *skew-symmetric matrices, and via the standard basis we identify each such matrix* $\hat{\xi}$ *by a vector* $\xi \in \mathbb{R}^3$, *this identification is known as the hat map*

$$\hat{\xi} = \begin{bmatrix} 0 & -\xi_3 & \xi_2 \\ \xi_3 & 0 & -\xi_1 \\ -\xi_2 & \xi_1 & 0 \end{bmatrix}. \tag{2.2.5}$$

Now, we also write the elements of $\mathfrak{so}(3)^*$ *as vectors in* $\mathbb{R}^3$ *with duality pairing* $\langle \mu, \xi \rangle = \mu^T \xi$. *With these representations, we find that the coadjoint action can be expressed as*

$$g \cdot \mu = \psi(g, \mu) = \mathrm{Ad}^*_{g^{-1}} \mu = g\mu,$$

the rightmost expression being a simple matrix-vector multiplication. Since g *is orthogonal, it follows that the coadjoint orbits foliate 3-space into spherical shells, and the coadjoint action is transitive on each of these orbits. The free rigid body can be cast as a problem on* $TSO(3)^*$ *with a left invariant Hamiltonian which reduces to the function*

$$H^-(\mu) = \frac{1}{2} \langle \mu, \mathbb{I}^{-1} \mu \rangle$$

on $\mathfrak{so}(3)^*$ *where* $\mathbb{I} : \mathfrak{so}(3) \to \mathfrak{so}(3)^*$ *is the inertia tensor. From this, we can now set* $f(\mu) = \partial H^- / \partial \mu = \mathbb{I}^{-1} \mu$. *We then recover the Euler free rigid body equation as*

$$\dot{\mu} = \psi_* (f(\mu)\big|_\mu = -\mathrm{ad}^*_{\mathbb{I}^{-1}\mu} \mu = -\mathbb{I}^{-1} \mu \times \mu,$$

where the last expression involves the cross product of vectors in $\mathbb{R}^3$.

2.2.2 Two classes of Lie group integrators

The simplest numerical integrator for linear spaces is the explicit Euler method. Given an initial value problem $\dot{y} = F(y)$, $y(0) = y_0$ the method is defined as $y_{n+1} = y_n + hF(y_n)$ for some stepsize h. In the spirit of the previous section, one could think of the Euler method as the h-flow of the constant vector field $F_{y_n}(y) = F(y_n)$, that is

$$y_{n+1} = \exp(hF_{y_n})\, y_n\,.$$

This definition of the Euler method makes sense also when F is replaced by a vector field on some manifold. In this general situation it is known as the Lie–Euler method.

We shall here consider the two classes of methods known as Runge–Kutta–Munthe–Kaas (RKMK) methods and Commutator-free Lie group methods.

For RKMK methods the underlying idea is to transform the problem from the manifold $\mathcal{M}$ to the Lie algebra $\mathfrak{g}$, take a time step, and map the result back to $\mathcal{M}$. The transformation we use is

$$y(t) = \exp(\sigma(t)) \cdot y_0, \quad \sigma(0) = 0.$$

The transformed differential equation for $\sigma(t)$ makes use of the derivative of the exponential mapping, the reader should consult [40] for details about the derivation, we give the final result

$$\dot{\sigma}(t) = \operatorname{dexp}^{-1}_{\sigma(t)}(f(\exp(\sigma(t)) \cdot y_0)). \tag{2.2.6}$$

The map $v \mapsto \operatorname{dexp}_u(v)$ is linear and invertible when u belongs to some sufficiently small neighborhood of $0 \in \mathfrak{g}$. It has an expansion in nested Lie brackets [21]. Using the operator $\operatorname{ad}_u(v) = [u, v]$ and its powers $\operatorname{ad}^2_u v = [u, [u, v]]$ etc, one can write

$$\operatorname{dexp}_u(v) = \left.\frac{e^z - 1}{z}\right|_{z=\operatorname{ad}_u}(v) = v + \frac{1}{2}[u, v] + \frac{1}{6}[u, [u, v]] + \cdots \tag{2.2.7}$$

and the inverse is

$$\operatorname{dexp}^{-1}_u(v) = \left.\frac{z}{e^z - 1}\right|_{z=\operatorname{ad}_u}(v) = v - \frac{1}{2}[u, v] + \frac{1}{12}[u, [u, v]] + \cdots \tag{2.2.8}$$

The RKMK methods are now obtained simply by applying some standard Runge–Kutta method to the transformed equation (2.2.6) with a time step h, using initial value $\sigma(0) = 0$. This leads to an output $\sigma_1 \in \mathfrak{g}$ and one simply sets $y_1 = \exp(\sigma_1) \cdot y_0$. Then one repeats the procedure replacing y_0 by y_1 in the next step etc. While solving (2.2.6) one needs to evaluate $\operatorname{dexp}^{-1}_u(v)$ as a part of the process. This can be done by truncating the series (2.2.8) since $\sigma(0) = 0$ implies that we always evaluate $\operatorname{dexp}^{-1}_u$ with $u = \mathcal{O}(h)$, and thus, the kth iterated commutator $\operatorname{ad}^k_u = \mathcal{O}(h^k)$. For a given Runge–Kutta method, there are some clever tricks that can be done to minimise the total number of commutators to be included from the expansion of $\operatorname{dexp}^{-1}_u v$, see [5, 41]. We give here one concrete example of an RKMK method proposed in [5]

$$\begin{aligned}
f_{n,1} &= hf(y_n),\\
f_{n,2} &= hf(\exp(\tfrac{1}{2} f_{n,1}) \cdot y_n),\\
f_{n,3} &= hf(\exp(\tfrac{1}{2} f_{n,2} - \tfrac{1}{8}[f_{n,1}, f_{n,2}]) \cdot y_n),\\
f_{n,4} &= hf(\exp(f_{n,3}) \cdot y_n),\\
y_{n+1} &= \exp(\tfrac{1}{6}(f_{n,1} + 2f_{n,2} + 2f_{n,3} + f_{n,4} - \tfrac{1}{2}[f_{n,1}, f_{n,4}])) \cdot y_n.
\end{aligned}$$

The other option is to compute the exact expression for $\operatorname{dexp}_u^{-1}(v)$ for the particular Lie algebra we use. For instance, it was shown in [8] that for the Lie algebra $\mathfrak{so}(3)$ one has

$$\operatorname{dexp}_u^{-1}(v) = v - \frac{1}{2}u \times v + \alpha^{-2}(1 - \tfrac{\alpha}{2}\cot\tfrac{\alpha}{2})\, u \times (u \times v).$$

We will present the corresponding formula for $\mathfrak{se}(3)$ in Section 2.2.3.

The second class of Lie group integrators to be considered here are the commutator-free methods, named this way in [6] to emphasize the contrast to RKMK schemes which usually include commutators in the method format. These schemes include the Crouch-Grossman methods [11] and they have the format

$$\begin{aligned} Y_{n,r} &= \exp\left(h\sum_k \alpha_{r,J}^k f_{n,k}\right)\cdots\exp\left(h\sum_k \alpha_{r,1}^k f_{n,k}\right)\cdot y_n, \\ f_{n,r} &= f(Y_{n,r}), \\ y_{n+1} &= \exp\left(h\sum_k \beta_J^k f_{n,k}\right)\cdots\exp\left(h\sum_k \beta_1^k f_{n,k}\right)\cdot y_n. \end{aligned}$$

Here the Runge–Kutta coefficients $\alpha_{r,j}^k$, β_j^r are related to a classical Runge–Kutta scheme with coefficients a_r^k, b_r in that $a_r^k = \sum_j \alpha_{r,j}^k$ and $b_r = \sum_j \beta_j^r$. The $\alpha_{r,j}^k$, β_j^r are usually chosen to obtain computationally inexpensive schemes with the highest possible order of convergence. The computational complexity of the above schemes depends on the cost of computing an exponential as well as of evaluating the vector field. Therefore it makes sense to keep the number of exponentials J in each stage as low as possible, and possibly also the number of stages s. A trick proposed in [6] was to select coefficients that make it possible to reuse exponentials from one stage to another. This is perhaps best illustrated through the following example from [6], a generalisation of the classical 4th order Runge–Kutta method.

$$\begin{aligned} Y_{n,1} &= y_n, \\ Y_{n,2} &= \exp(\tfrac{1}{2}hf_{n,1})\cdot y_n, \\ Y_{n,3} &= \exp(\tfrac{1}{2}hf_{n,2})\cdot y_n, \\ Y_{n,4} &= \exp(hf_{n,3} - \tfrac{1}{2}hf_{n,1})\cdot Y_{n,2}, \\ y_{n+\frac{1}{2}} &= \exp(\tfrac{1}{12}h(3f_{n,1} + 2f_{n,2} + 2f_{n,3} - f_{n,4}))\cdot y_n, \\ y_{n+1} &= \exp(\tfrac{1}{12}h(-f_{n,1} + 2f_{n,2} + 2f_{n,3} + 3f_{n,4}))\cdot y_{n+\frac{1}{2}}, \end{aligned} \tag{2.2.9}$$

where $f_{n,i} = f(Y_{n,i})$. Here, we see that one exponential is saved in computing $Y_{n,4}$ by making use of $Y_{n,2}$.

2.2.3 An exact expression for $\operatorname{dexp}_u^{-1}(v)$ in $\mathfrak{se}(3)$

As an alternative to using a truncated version of the infinite series for $\operatorname{dexp}_u^{-1}$ (2.2.8), one can consider exact expressions obtained for certain Lie algebras. Since $\mathfrak{se}(3)$ is particularly important in applications to mechanics, we give here its exact expression. For this, we represent elements of $\mathfrak{se}(3)$ as a pair $(A, a) \in \mathbb{R}^3 \times \mathbb{R}^3 \cong \mathbb{R}^6$, the first component corresponding to a skew-symmetric matrix $\hat{A}$ via (2.2.5) and a is the translational part. Now, let $\varphi(z)$ be a real analytic function at $z = 0$. We define

$$\varphi_+(z) = \frac{\varphi(iz) + \varphi(-iz)}{2}, \qquad \varphi_-(z) = \frac{\varphi(iz) - \varphi(-iz)}{2i}.$$

We next define the four functions

$$g_1(z) = \frac{\varphi_-(z)}{z},\ \tilde{g}_1(z) = \frac{g_1'(z)}{z}, \quad g_2(z) = \frac{\varphi(0) - \varphi_+(z)}{z^2},\ \tilde{g}_2(z) = \frac{g_2'(z)}{z},$$

and the two scalars $\rho = A^T a$, $\alpha = \|A\|_2$. One can show that for any (A, a) and (B, b) in $\mathfrak{se}(3)$, it holds that

$$\varphi(\operatorname{ad}_{(A,a)})(B, b) = (C, c),$$

where

$$\begin{aligned} C &= \varphi(0)B + g_1(\alpha)\, A \times B + g_2(\alpha)\, A \times (A \times B), \\ c &= \varphi(0)b + g_1(\alpha)\,(a \times B + A \times b) + \rho\tilde{g}_1(\alpha)\, A \times B + \rho\tilde{g}_2(\alpha)\, A \times (A \times B) \\ &\quad + g_2(\alpha)\,(a \times (A \times B) + A \times (a \times B) + A \times (A \times b)). \end{aligned}$$

Considering for instance (2.2.8), we may now use $\varphi(z) = \frac{z}{e^z - 1}$ to calculate

$$g_1(z) = -\frac{1}{2},\ \tilde{g}_1(z) = 0,\ g_2(z) = \frac{1 - \frac{z}{2}\cot\frac{z}{2}}{z^2},\ \tilde{g}_2(z) = \frac{1}{z}\frac{d}{dz}g_2(z),\ \varphi(0) = 1.$$

and thereby obtain an expression for $\operatorname{dexp}^{-1}_{(A,a)}(B, b)$ with the formula above.

Similar types of formulas are known for computing the matrix exponential as well as functions of the ad-operator for several other Lie groups of small and medium dimension. For instance in [38] a variety of coordinate mappings for rigid body motions are discussed. For Lie algebras of larger dimension, both the exponential mapping and $\operatorname{dexp}_u^{-1}$ may become computationally infeasible. For these cases, one may benefit from replacing the exponential by some other coordinate map for the Lie group $\phi : \mathfrak{g} \to G$. One option is to use canonical coordinates of the second kind [45]. Then for some Lie groups such as the orthogonal, unitary and symplectic groups, there exist other maps that can be used and which are computationally less expensive. A popular choice is the Cayley transformation [13].

2.3 Hamiltonian systems on Lie groups

In this section we consider Hamiltonian systems on Lie groups. These systems (and their Lagrangian counterpart) often appear in mechanics applications as building blocks for more realistic systems with additional damping and control forces. We consider canonical systems on the cotangent bundle of a Lie group and Lie-Poisson systems which can arise by symmetry reduction or otherwise. We illustrate the various cases with different formulations of the heavy top system.

2.3.1 Semi-direct products

The coadjoint action by G on $\mathfrak{g}^*$ is denoted Ad^*_g defined for any $g \in G$ as

$$\langle \mathrm{Ad}^*_g \mu, \xi \rangle = \langle \mu, \mathrm{Ad}_g \xi \rangle, \quad \forall \xi \in \mathfrak{g}, \tag{2.3.1}$$

where $\mathrm{Ad} : \mathfrak{g} \to \mathfrak{g}$ is the adjoint representation and for a duality pairing $\langle \cdot, \cdot \rangle$ between $\mathfrak{g}^*$ and $\mathfrak{g}$.

We consider the cotangent bundle of a Lie group G, T^*G and identify it with $G \times \mathfrak{g}^*$ using the right multiplication $R_g : G \to G$ and its tangent mapping $R_{g*} := TR_g$. The cartesian product $G \times \mathfrak{g}^*$ can be given a semi-direct product structure that turns it into a Lie group $\mathbf{G} := G \ltimes \mathfrak{g}^*$ where the group multiplication is

$$(g_1, \mu_1) \cdot (g_2, \mu_2) = (g_1 \cdot g_2, \mu_1 + \mathrm{Ad}^*_{g_1^{-1}} \mu_2). \tag{2.3.2}$$

Acting by left multiplication any vector field $F \in \mathcal{X}(\mathbf{G})$ is expressed by means of a map $f : \mathbf{G} \to T_e\mathbf{G}$,

$$F(g, \mu) = T_e R_{(g,\mu)} f(g, \mu) = (R_{g*} f_1, f_2 - \mathrm{ad}^*_{f_1} \mu), \tag{2.3.3}$$

where $f_1 = f_1(g, \mu) \in \mathfrak{g}$, $f_2 = f_2(g, \mu) \in \mathfrak{g}^*$ are the two components of f.

2.3.2 Symplectic form and Hamiltonian vector fields

The right trivialised[2] symplectic form pulled back to $\mathbf{G}$ reads

$$\begin{aligned} &\omega_{(g,\mu)}((R_{g*}\xi_1, \delta\nu_1), (R_{g*}\xi_2, \delta\nu_2)) \\ &\qquad = \langle \delta\nu_2, \xi_1 \rangle + - \langle \delta\nu_1, \xi_2 \rangle - \langle \mu, [\xi_1, \xi_2] \rangle, \qquad \xi_1, \xi_2 \in \mathfrak{g}. \end{aligned} \tag{2.3.4}$$

[2] $\omega_{(g,\mu)}$ is obtained from the natural symplectic form on T^*G (which is a differential two-form), defined as

$$\Omega_{(g,p_g)}((\delta\nu_1, \delta\pi_1), (\delta\nu_2, \delta\pi_2)) = \langle \delta\pi_2, \delta\nu_1 \rangle - \langle \delta\pi_1, \delta\nu_2 \rangle,$$

by right trivialization.

See [31] for more details, proofs and for a the left trivialized symplectic form. The vector field F is a Hamiltonian vector field if it satisfies

$$\mathrm{i}_F\omega = dH,$$

for some Hamiltonian function $H : T^*G \to \mathbb{R}$, where i_F is defined as $\mathrm{i}_F(X) := \omega(F, X)$ for any vector field X. This implies that the map f for such a Hamiltonian vector field gets the form

$$f(g,\mu) = \left(\frac{\partial H}{\partial \mu}(g,\mu), -R_g^*\frac{\partial H}{\partial g}(g,\mu)\right). \tag{2.3.5}$$

The following is a one-parameter family of symplectic Lie group integrators on T^*G:

$$M_\theta = \mathrm{dexp}^*_{-\xi}(\mu_0 + \mathrm{Ad}^*_{\exp(\theta\xi)}(\bar{n})) - \theta\mathrm{dexp}^*_{-\theta\xi}\mathrm{Ad}^*_{\exp(\theta\xi)}(\bar{n}), \tag{2.3.6}$$

$$(\xi, \bar{n}) = hf\big(\exp(\theta\xi)\cdot g_0, M_\theta\big), \tag{2.3.7}$$

$$(g_1, \mu_1) = (\exp(\xi), \mathrm{Ad}^*_{\exp((\theta-1)\xi)}\bar{n})\cdot(g_0, \mu_0). \tag{2.3.8}$$

For higher order integrators of this type and a complete treatment see [3].

2.3.3 Reduced equations Lie Poisson systems

A mechanical system formulated on the cotangent bundle T^*G with a left or right invariant Hamiltonian can be reduced to a system on $\mathfrak{g}^*$ [33]. In fact for a Hamiltonian H right invariant under the left action of G, $\frac{\partial H}{\partial g} = 0$, and from (2.3.3) and (2.3.5) we get for the second equation

$$\dot{\mu} = \mp\mathrm{ad}^*_{\frac{\partial H}{\partial \mu}}\mu, \tag{2.3.9}$$

where the positive sign is used in case of left invariance (see e.g. section 13.4 in [35]). The solution to this system preserves coadjoint orbits, thus using the Lie group action

$$g\cdot\mu = \mathrm{Ad}^*_{g^{-1}}\mu,$$

to build a Lie group integrator results in preservation of such coadjoint orbits. Lie group integrators for this interesting case were studied in [15].

The Lagrangian counterpart to these Hamiltonian equations are the Euler–Poincaré equations[3], [24].

[3]The Euler–Poincaré equations are Euler–Lagrange equations with respect to a Lagrange–d'Alembert principle obtained taking constraint variations.

2.3.4 Three different formulations of the heavy top equations

The heavy top is a simple test example for illustrating the behaviour of Lie group methods. We will consider three different formulations for this mechanical system. The first formulation is on $T^*SO(3)$ where the equations are canonical Hamiltonian, a second point of view is that the system is a Lie–Poisson system on $\mathfrak{se}(3)^*$, and finally it is canonical Hamiltonian on a larger group with a quadratic Hamiltonian function. The three different formulations suggest the use of different Lie group integrators.

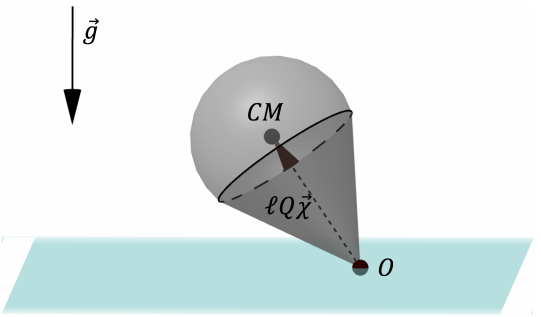


Figure 2.1: Illustration of the heavy top, where CM is the center of mass of the body, O is the fixed point, $\vec{g}$ is the gravitational acceleration vector, and $\ell, Q, \vec{\chi}$ follow the notation introduced in Section 2.3.4

Heavy top equations on $T^*SO(3)$.

The heavy top is a rigid body with a fixed point in a gravitational field. The phase space of this mechanical system is $T^*SO(3)$ where the equations of the heavy top are in canonical Hamiltonian form. Assuming (Q, p) are coordinates for $T^*SO(3)$, $\Pi = (T_eL_Q)^*(p)$ is the left trivialized or body momentum. The Hamiltonian of the heavy top is given in terms of (Q, Π) as

$$H\colon SO(3) \ltimes \mathfrak{so}(3)^* \to \mathbb{R}, \quad H(Q,\Pi) = \frac{1}{2}\langle \Pi, \mathbb{I}^{-1}\Pi\rangle + Mg\ell\,\Gamma\cdot\mathcal{X}, \quad \Gamma = Q^{-1}\Gamma_0,$$

where $\mathbb{I}\colon \mathfrak{so}(3) \to \mathfrak{so}(3)^*$ is the inertia tensor, here represented as a diagonal 3×3 matrix, $\Gamma = Q^{-1}\Gamma_0$, where $\Gamma_0 \in \mathbb{R}^3$ is the axis of the spatial coordinate system parallel to the direction of gravity but pointing upwards, M is the mass of the body, g is the gravitational acceleration, $\mathcal{X}$ is the body fixed unit vector of the oriented line segment pointing from the fixed point to the center of mass of the body, ℓ is the length of this segment. The equations of motion on $SO(3) \ltimes \mathfrak{so}(3)^*$

are

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi + Mg\ell\,\Gamma \times \mathcal{X}, \tag{2.3.10}$$

$$\dot{Q} = Q\widehat{\mathbb{I}^{-1}\Pi}. \tag{2.3.11}$$

The identification of $T^*SO(3)$ with $SO(3) \ltimes \mathfrak{so}(3)^*$ via right trivialization leads to the spatial momentum variable $\pi = (T_e R_Q)^*(p) = Q\Pi$. The equations written in the space variables (Q, π) get the form

$$\dot{\pi} = Mg\ell\,\Gamma_0 \times Q\mathcal{X}, \tag{2.3.12}$$

$$\dot{Q} = \hat{\omega}Q \qquad \omega = Q\mathbb{I}^{-1}Q^T\pi. \tag{2.3.13}$$

where, the first equation states that the component of π parallel to Γ_0 is constant in time. These equations can be obtained from (2.3.3) and (2.3.5) on the right trivialized $T^*SO(3)$, $SO(3) \ltimes \mathfrak{so}(3)^*$, with the heavy top Hamiltonian and the symplectic Lie group integrators (2.3.7)-(2.3.8) can be applied in this case. Similar methods were proposed in [31] and [48].

Heavy top equations on $\mathfrak{se}^*(3)$.

The Hamiltonian of the heavy top is not invariant under the action of $SO(3)$, so the equations (2.3.10)-(2.3.11) given in section (2.3.4) cannot be reduced to $\mathfrak{so}^*(3)$, nevertheless the heavy top equations are Lie–Poisson on $\mathfrak{se}^*(3)$, [17, 47, 52].

Observe that the equations of the heavy top on $T^*SO(3)$ (2.3.10)-(2.3.11) can be easily modified eliminating the variable $Q \in SO(3)$ and replacing it with $\Gamma \in \mathbb{R}^3$ $\Gamma = Q^{-1}\Gamma_0$ to obtain

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi + Mg\ell\,\Gamma \times \mathcal{X}, \tag{2.3.14}$$

$$\dot{\Gamma} = \Gamma \times (\mathbb{I}^{-1}\Pi). \tag{2.3.15}$$

We will see that the solutions of these equations evolve on $\mathfrak{se}^*(3)$. In what follows, we consider elements of $\mathfrak{se}^*(3)$ to be pairs of vectors in $\mathbb{R}^3$, e.g. (Π, Γ). Correspondingly the elements of $SE(3)$ are represented as pairs $(g, \mathbf{u})$ with $g \in SO(3)$ and $\mathbf{u} \in \mathbb{R}^3$. The group multiplication in $SE(3)$ is then

$$(g_1, \mathbf{u}_1) \cdot (g_2, \mathbf{u}_2) = (g_1 g_2, g_1\mathbf{u}_2 + \mathbf{u}_1),$$

where $g_1 g_2$ is the product in $SO(3)$ and $g_1\mathbf{u}$ is the product of a 3×3 orthogonal matrix with a vector in $\mathbb{R}^3$. The coadjoint representation and its infinitesimal generator on $\mathfrak{se}^*(3)$ take the form

$$\mathrm{Ad}^*_{(g,\mathbf{u})}(\Pi, \Gamma) = (g^{-1}(\Pi - \mathbf{u} \times \Gamma), g^{-1}\Gamma), \;\; \mathrm{ad}^*_{(\xi,\mathbf{u})}(\Pi, \Gamma) = (-\xi \times \Pi - \mathbf{u} \times \Gamma, -\xi \times \Gamma).$$

Using this expression for $\mathrm{ad}^*_{(\xi,\mathbf{u})}$ with $(\xi = \frac{\partial H}{\partial \Pi}, \mathbf{u} = \frac{\partial H}{\partial \Gamma})$, it can be easily seen that the equations (2.3.9) in this setting reproduce the heavy top equations (2.3.14)-(2.3.15). Therefore the equations are Lie–Poisson equations on $\mathfrak{se}^*(3)$. However since the heavy top is a rigid body with a fixed point and there are no translations, these equations do not arise from a reduction of $T^*SE(3)$. Moreover the Hamiltonian on $\mathfrak{se}(3)^*$ is not quadratic and the equations are not geodesic equations. Implicit and explicit Lie group integrators applicable to this formulation of the heavy top equations and preserving coadjoint orbits were discussed in [15], for a variable stepsize integrator applied to this formulation of the heavy top see [12].

Heavy top equations with quadratic Hamiltonian.

We rewrite the heavy top equations one more time considering the constant vector $\mathbf{p} = -Mg\ell\mathcal{X}$ as a momentum variable conjugate to the position $\mathbf{q} \in \mathbb{R}^3$ and where $\mathbf{p} = Q^{-1}\Gamma_0 + \dot{\mathbf{q}}$, and the Hamiltonian is a quadratic function of Π, Q, $\mathbf{p}$ and $\mathbf{q}$:

$$H\colon T^*SO(3) \times {\mathbb{R}^3}^* \times \mathbb{R}^3 \to \mathbb{R},$$
$$H((\Pi, Q), (\mathbf{p}, \mathbf{q})) = \frac{1}{2}\langle \Pi, \mathbb{I}^{-1}\Pi\rangle + \frac{1}{2}\|\mathbf{p} - Q^{-1}\Gamma_0\|^2 - \frac{1}{2}\|Q^{-1}\Gamma_0\|^2,$$

see [23, section 8.5]. This Hamiltonian is invariant under the left action of $SO(3)$. The corresponding equations are canonical on $T^*S \equiv S \ltimes \mathfrak{s}^*$ where $S = SO(3) \times \mathbb{R}^3$ with Lie algebra $\mathfrak{s} := \mathfrak{so}(3) \times \mathbb{R}^3$ and T^*S can be identified with $T^*SO(3) \times {\mathbb{R}^3}^* \times \mathbb{R}^3$. The equations are

$$\dot{\Pi} = \Pi \times \mathbb{I}^{-1}\Pi - (Q^{-1}\Gamma_0) \times \mathbf{p}, \tag{2.3.16}$$
$$\dot{Q} = Q\widehat{\mathbb{I}^{-1}\Pi}, \tag{2.3.17}$$
$$\dot{\mathbf{p}} = \mathbf{0}, \tag{2.3.18}$$
$$\dot{\mathbf{q}} = \mathbf{p} - Q^{-1}\Gamma_0. \tag{2.3.19}$$

and in the spatial momentum variables

$$\dot{\pi} = -\Gamma_0 \times Q\mathbf{p}, \tag{2.3.20}$$
$$\dot{Q} = \hat{\omega}Q, \quad \omega = Q\mathbb{I}^{-1}Q^T\pi, \tag{2.3.21}$$
$$\dot{\mathbf{p}} = \mathbf{0}, \tag{2.3.22}$$
$$\dot{\mathbf{q}} = \mathbf{p} - Q^{-1}\Gamma_0. \tag{2.3.23}$$

Similar formulations were considered in [30] for the stability analysis of an underwater vehicle. A similar but different formulation of the heavy top was considered in [4].

Numerical experiments.

We apply various implicit Lie group integrators to the heavy top system. The test problem we consider is the same as in [4], where $Q(0) = I$, $\ell = 2$, $M = 15$ $\mathbb{I} = \mathrm{diag}(0.234375, 0.46875, 0.234375)$, $\pi(0) = \mathbb{I}(0, 150, -4.61538)$, $\mathcal{X} = (0, 1, 0)$ $\Gamma_0 = (0, 0, -9.81)$.

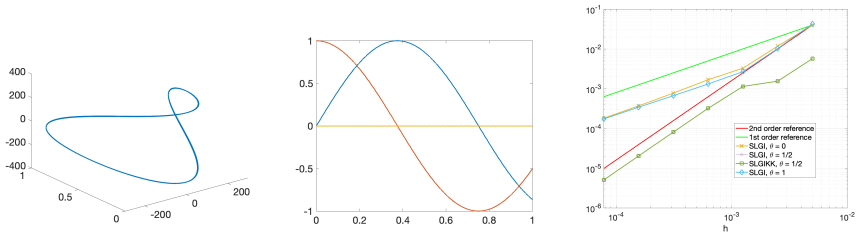


Figure 2.2: Symplectic Lie group integrators integration on the time interval $[0, 1]$. Left: 3D plot of $M\ell Q^{-1}\Gamma_0$. Center: components of $Q\mathcal{X}$. The left and center plots are computed with the same step-size. Right: verification of the order of the methods.

In Figure 2.2 we report the performance of the symplectic Lie group integrators (2.3.6)-(2.3.8) applied both on the equations (2.3.12)-(2.3.13) with $\theta = 0$, $\theta = \frac{1}{2}$ and $\theta = 1$ (SLGI), and to the equations (2.3.20)-(2.3.23) with $\theta = \frac{1}{2}$ (SLGIKK). The methods with $\theta = \frac{1}{2}$ attain order 2. In Figure 2.3 we show the energy error for the symplectic Lie group integrators with $\theta = \frac{1}{2}$ and $\theta = 0$ integrating with stepsize $h = 0.01$ for 6000 steps.

2.4 Variable step size

One approach for varying the step size is based on the use of an embedded Runge–Kutta pair. This principle can be carried from standard Runge–Kutta methods in vector spaces to the present situation with RKMK and commutator-free schemes via minor modifications. We briefly summarise the main principle of embedded pairs before giving more specific details for the case of Lie group integrators. This approach is very well documented in the literature and goes back to Merson [36] and a detailed treatment can be found in [19, p. 165–168].

An embedded pair consists of a main method used to propagate the numerical solution, together with some auxiliary method that is only used to obtain an estimate of the local error. This local error estimate is in turn used to derive a step size adjustment formula that attempts to keep the local error estimate approximately equal to some user defined tolerance tol in every step. Suppose

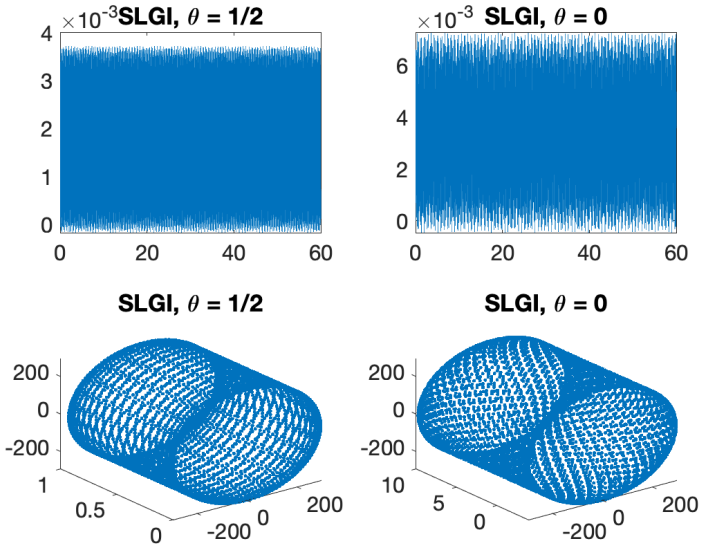


Figure 2.3: Symplectic Lie group integrators, long time integration, $h = 0.01$, 6000 steps.. Top: energy error, bottom 3D plot of $M\ell Q^{-1}\Gamma_0$.

the main method is of order p and the auxiliary method is of order $\tilde{p} \neq p$. [4] Both methods are applied to the input value y_n and yields approximations y_{n+1} and $\tilde{y}_{n+1}$ respectively, using the same step size h_{n+1}. Now, some distance measure[5] between y_{n+1} and $\tilde{y}_{n+1}$ provides an estimate e_{n+1} for the size of the local truncation error. Thus, $e_{n+1} = Ch_{n+1}^{\tilde{p}+1} + \mathcal{O}(h^{\tilde{p}+2})$. Aiming at $e_{n+1} \approx \text{tol}$ in every step, one may use a formula of the type

$$h_{n+1} = \theta \left(\frac{\text{tol}}{e_{n+1}} \right)^{\frac{1}{\tilde{p}+1}} h_n \tag{2.4.1}$$

where θ is a 'safety factor', typically chosen between 0.8 and 0.9. In case the step is rejected because $e_n > \text{tol}$ we can redo the step with a step size obtained by the same formula. We summarise the approach in the following algorithm

Given y_n, h_n, tol
Let $h := h_n$

[4] In this paper we will assume $\tilde{p} < p$ in which case the local error estimate is relevant for the approximation $\tilde{y}_{n+1}$

[5] There are many options for how to do this in practice, and the choice may also depend on the application. E.g. a Riemannian metric is a natural and robust alternative here.

repeat
    Compute y_{n+1}, $\tilde{y}_{n+1}$, e_{n+1} from y_n, h
    Update stepsize $h := \theta \left(\frac{\text{tol}}{e_{n+1}}\right)^{\alpha} h$
    accepted := $e_{n+1} < \text{tol}$
    if accepted
        update step index: $n := n+1$
        $h_n := h$
until accepted

Here we have used again the safety factor θ, and the parameter α is generally chosen as $\alpha = \frac{1}{1+\min(p,\tilde{p})}$.

2.4.1 RKMK methods with variable stepsize

We need to specify how to calculate the quantity e_{n+1} in each step. For RKMK methods the situation is simplified by the fact that we are solving the local problem (2.2.6) in the linear space $\mathfrak{g}$, where the known theory can be applied directly. So any standard embedded pair of Runge–Kutta methods described by coefficients $(a_{ij}, b_i, \tilde{a}_{ij}, \tilde{b}_i)$ of orders $(p, \tilde{p})$ can be applied to the full dexpinv-equation (2.2.6) to obtain local Lie algebra approximations σ_1, $\tilde{\sigma}_1$ and one uses e.g. $e_{n+1} = \|\sigma_1 - \tilde{\sigma}_1\|$ (note that the equation itself depends on y_n). For methods which use a truncated version of the series for dexp_u^{-1} one may also try to optimise performance by including commutators that are shared between the main method and the auxiliary scheme.

2.4.2 Commutator-free methods with variable stepsize

For the commutator-free methods of section 2.2.2 the situation is different since such methods do not have a natural local representation in a linear space. One can still derive embedded pairs, and this can be achieved by studying order conditions [43] as was done in [12]. Now one obtains after each step two approximations y_{n+1} and $\tilde{y}_{n+1}$ on $\mathcal{M}$ both by using the same initial value y_n and step size h_n. One must also have access to some metric d to calculate $e_{n+1} = d(y_{n+1}, \tilde{y}_{n+1})$ We give a few examples of embedded pairs.

Pairs of order $(p, \tilde{p}) = (3, 2)$

It is possible to obtain embedded pairs of order 3(2) which satisfy the requirements above. We present two examples from [12]. The first one reuses the

second stage exponential in the update

$$\begin{aligned}
Y_{n,1} &= y_n, \\
Y_{n,2} &= \exp(\tfrac{1}{3}hf_{n,1}) \cdot y_n, \\
Y_{n,3} &= \exp(\tfrac{2}{3}hf_{n,2}) \cdot y_n, \\
y_{n+1} &= \exp(h(-\tfrac{1}{12}f_{n,1} + \tfrac{3}{4}f_{n,3})) \cdot Y_{n,2}, \\
\tilde{y}_{n+1} &= \exp(\tfrac{1}{2}h(f_{n,2} + f_{n,3})) \cdot y_n.
\end{aligned}$$

One could also have reused the third stage $Y_{n,3}$ in the update, rather than $Y_{n,2}$.

$$\begin{aligned}
Y_{n,1} &= y_n, \\
Y_{n,2} &= \exp(\tfrac{2}{3}hf_{n,1}) \cdot y_n, \\
Y_{n,3} &= \exp(h(\tfrac{5}{12}f_{n,1} + \tfrac{1}{4}f_{n,2}) \cdot y_n, \\
y_{n+1} &= \exp(h(-\tfrac{1}{6}f_{n,1} - \tfrac{1}{2}f_{n,2} + f_{n,3})) \cdot Y_{n,3}, \\
\tilde{y}_{n+1} &= \exp(\tfrac{1}{4}h(f_{n,1} + 3f_{n,3})) \cdot y_n.
\end{aligned}$$

It is always understood that the frozen vector fields are $f_{n,i} := f_{Y_{n,i}}$.

Order $(4,3)$

The procedure of deriving efficient pairs becomes more complicated as the order increases. In [12] a low cost pair of order $(4,3)$ was derived, in the sense that one attempted to minimise the number of stages and exponentials in the embedded pair as a whole. This came, however, at the expense of a relatively large error constant. So rather than presenting the method from that paper, we suggest a simpler procedure at the cost of some more computational work per step, we simply furnish the commutator-free method of section 2.2 by a third order auxiliary scheme. It can be described as follows:

1. Compute $Y_{n,i}$, $i = 1\ldots,4$ and y_{n+1} from (2.2.9)
2. Compute an additional stage $\bar{Y}_{n,3}$ and then $\tilde{y}_{n+1}$ as

$$\begin{aligned}
\bar{Y}_{n,3} &= \exp(\tfrac{3}{4}hf_{n,2}) \cdot y_n, \\
\tilde{y}_{n+1} &= \exp(\tfrac{h}{9}(-f_{n,1} + 3f_{n,2} + 4\bar{f}_{n,3})) \cdot \exp(\tfrac{h}{3}f_{n,1}) \cdot y_n.
\end{aligned} \tag{2.4.2}$$

2.5 The N-fold 3D pendulum

In this section, we present a model for a system of N connected 3-dimensional pendulums. The modelling part comes from [28], and here we study the vector field describing the dynamics, in order to re-frame it into the Lie group

integrators setting described in the previous sections. The model we use is not completely realistic since, for example, it neglects possible interactions between pendulums, and it assumes ideal spherical joints between them. However, this is still a relevant example from the point of view of geometric numerical integration. More precisely, we show a possible way to work with a configuration manifold which is not a Lie group, applying the theoretical instruments introduced before. The Lagrangian we consider is a function from $(TS^2)^N$ to $\mathbb{R}$.

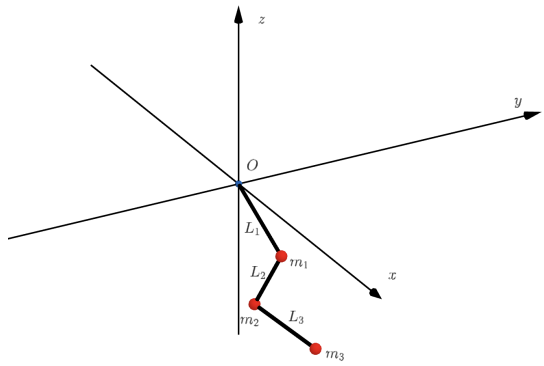


Figure 2.4: 3−fold pendulum at a fixed time instant, with fixed point placed at the origin.

Instead of the coordinates $(q_1, ..., q_N, \dot{q}_1, ..., \dot{q}_N)$, where $\dot{q}_i \in T_{q_i}S^2$, we choose to work with the angular velocities. Precisely,

$$T_{q_i}S^2 = \{v \in \mathbb{R}^3 : v^T q_i = 0\} = \langle q_i \rangle^\perp \subset \mathbb{R}^3,$$

and hence for any $\dot{q}_i \in T_{q_i}S^2$ there exist $\omega_i \in \mathbb{R}^3$ such that $\dot{q}_i = \omega_i \times q_i$, which can be interpreted as the angular velocity of q_i. So we can assume without loss of generality that $\omega_i^T q_i = 0$ (i.e. $\omega_i \in T_{q_i}S^2$) and pass to the coordinates $(q_1, \omega_1, q_2, \omega_2, ..., q_N, \omega_N) \in (TS^2)^N$ to describe the dynamics. In this section we denote with $m_1, ..., m_N$ the masses of the pendulums and with $L_1, ..., L_N$ their lengths. Figure 2.4 shows the case $N = 3$. We organize the section into three parts:

1. We define the transitive Lie group action used to integrate this model numerically,

2. We show a possible way to express the dynamics in terms of the infinitesimal generator of this action, for the general case of N joint pendulums,

3. We focus on the case $N = 2$, as a particular example. For this setting, we present some numerical experiment comparing various Lie group integrators and some classical numerical integrator. Then we conclude with numerical experiments on variable step size.

2.5.1 Transitive group action on $(TS^2)^N$

We characterize a transitive action for $(TS^2)^N$, starting with the case $N = 1$ and generalizing it to $N > 1$. The action we consider is based on the identification between $\mathfrak{se}(3)$, the Lie algebra of $SE(3)$, and $\mathbb{R}^6$. We start from the Ad-action of $SE(3)$ on $\mathfrak{se}(3)$ (see [23]), which writes

$$\mathrm{Ad} : SE(3) \times \mathfrak{se}(3) \to \mathfrak{se}(3),$$

$$\mathrm{Ad}((R, r), (u, v)) = (Ru, Rv + \hat{r}Ru).$$

Since $\mathfrak{se}(3) \simeq \mathbb{R}^6$, the Ad-action allows us to define the following Lie group action on $\mathbb{R}^6$

$$\psi : SE(3) \times \mathbb{R}^6 \to \mathbb{R}^6, \ \psi((R, r), (u, v)) = (Ru, Rv + \hat{r}Ru).$$

We can think of ψ as a Lie group action on TS^2 since, for any $q \in \mathbb{R}^3$, it maps points of

$$TS^2_{|q|} := \{(\tilde{q}, \tilde{\omega}) \in \mathbb{R}^3 \times \mathbb{R}^3 : \ \tilde{\omega}^T \tilde{q} = 0, \ |\tilde{q}| = |q|\} \subset \mathbb{R}^6$$

into other points of $TS^2_{|q|}$. Moreover, with standard arguments (see [42]), it is possible to prove that the orbit of a generic point $m = (q, \omega) \in \mathbb{R}^6$ with $\omega^T q = 0$ coincides with

$$\mathrm{Orb}(m) = TS^2_{|q|}.$$

In particular, when $q \in \mathbb{R}^3$ is a unit vector (i.e. $q \in S^2$), ψ allows us to define a transitive Lie group action on $TS^2 = TS^2_{|q|=1}$ which writes

$$\psi : SE(3) \times TS^2 \to TS^2,$$

$$\psi((A, a), (q, \omega)) := \psi_{(A,a)}(q, \omega) = (Aq, A\omega + \hat{a}Aq) = (\bar{q}, \bar{\omega}).$$

To conclude the description of the action, we report here its infinitesimal generator which is fundamental in the Lie group integrators setting

$$\psi_*((u, v))\big|_{(q,\omega)} = (\hat{u}q, \hat{u}\omega + \hat{v}q).$$

We can extend this construction to the case $N > 1$ in a natural way, i.e. through the action of a Lie group obtained from cartesian products of $SE(3)$ and equipped with the direct product structure. More precisely, we consider the group $G = (SE(3))^N$ and by direct product structure we mean that for any pair of elements

$$\delta^{(1)} = (\delta^{(1)}_1, ..., \delta^{(1)}_N), \quad \delta^{(2)} = (\delta^{(2)}_1, ..., \delta^{(2)}_N) \in G,$$

denoted with $*$ the semidirect product of $SE(3)$, we define the product $\circ$ on G as

$$\delta^{(1)} \circ \delta^{(2)} := (\delta^{(1)}_1 * \delta^{(2)}_1, ..., \delta^{(1)}_N * \delta^{(2)}_N) \in G.$$

With this group structure defined, we can generalize the action introduced for $N = 1$ to larger Ns as follows

$$\psi : (SE(3))^N \times (TS^2)^N \to (TS^2)^N,$$

$$\psi((A_1, a_1, ..., A_N, a_n), (q_1, \omega_1, ..., q_N, \omega_N)) =$$
$$= (A_1 q_1, A_1\omega_1 + \hat{a}_1 A_1 q_1, ..., A_N q_N, A_N\omega_N + \hat{a}_N A_N q_N),$$

whose infinitesimal generator writes

$$\psi_*(\xi)|_m = (\hat{u}_1 q_1, \hat{u}_1\omega_1 + \hat{v}_1 q_1, ..., \hat{u}_N q_N, \hat{u}_N\omega_N + \hat{v}_N q_N),$$

where $\xi = [u_1, v_1, ..., u_N, v_N] \in \mathfrak{se}(3)^N$ and $m = (q_1, \omega_1, ..., q_N, \omega_N) \in (TS^2)^N$. We have now the only group action we need to deal with the $N-$fold spherical pendulum. In the following part of this section we work on the vector field describing the dynamics and adapt it to the Lie group integrators setting.

2.5.2 Full chain

We consider the vector field $F \in \mathfrak{X}((TS^2)^N)$, describing the dynamics of the N-fold 3D pendulum, and we express it in terms of the infinitesimal generator of the action defined above. More precisely, we find a function $F : (TS^2)^N \to \mathfrak{se}(3)^N$ such that

$$\psi_*(f(m))|_m = F|_m, \ \forall m \in (TS^2)^N.$$

We omit the derivation of F starting from the Lagrangian of the system, which can be found in the section devoted to mechanical systems on $(S^2)^N$ of [28]. The configuration manifold of the system is $(S^2)^N$, while the Lagrangian, expressed in terms of the variables $(q_1, \omega_1, ..., q_N, \omega_N) \in (TS^2)^N$, writes

$$L(q,\omega) = T(q,\omega) - U(q) = \frac{1}{2}\sum_{i,j=1}^{N}\Big(M_{ij}\omega_i^T\hat{q}_i^T\hat{q}_j\omega_j\Big) - \sum_{i=1}^{N}\Big(\sum_{j=i}^{N} m_j\Big)gL_i e_3^T q_i,$$

where

$$M_{ij} = \Big(\sum_{k=\max\{i,j\}}^{N} m_k\Big)L_i L_j I_3 \in \mathbb{R}^{3\times 3}$$

is the inertia matrix of the system, I_3 is the 3×3 identity matrix, and $e_3 = [0,0,1]^T$. Noticing that when $i = j$ we get

$$\omega_i^T\hat{q}_i^T\hat{q}_i\omega_i = \omega_i^T(I_3 - q_i q_i^T)\omega_i = \omega_i^T\omega_i,$$

we simplify the notation writing

$$T(q,\omega) = \frac{1}{2}\sum_{i,j=1}^{N}\Big(\omega_i^T R(q)_{ij}\omega_j\Big),$$

where $R(q) \in \mathbb{R}^{3N\times 3N}$ is a symmetric block matrix defined as

$$R(q)_{ii} = \Big(\sum_{j=i}^{N} m_j\Big) L_i^2 I_3 \in \mathbb{R}^{3\times 3},$$

$$R(q)_{ij} = \Big(\sum_{k=j}^{N} m_k\Big) L_i L_j \hat{q}_i^T \hat{q}_j \in \mathbb{R}^{3\times 3} = R(q)_{ji}^T,\ i < j.$$

The vector field on which we need to work defines the following first-order ODE

$$\dot{q}_i = \omega_i \times q_i,\ i = 1, ..., N,$$

$$R(q)\dot{\omega} = \left[\sum_{\substack{j=1\\ j\neq i}}^{N} M_{ij}|\omega_j|^2 \hat{q}_i q_j - \Big(\sum_{j=i}^{N} m_j\Big) g L_i \hat{q}_i e_3\right]_{i=1,...,N} \in \mathbb{R}^{3N}.$$

By direct computation it is possible to see that, for any $q = (q_1, ..., q_N) \in (S^2)^N$ and $\omega \in T_{q_1}S^2 \times ... \times T_{q_N}S^2$, we have

$$(R(q)\omega)_i \in T_{q_i}S^2.$$

Therefore, there is a well-defined linear map

$$A_q : T_{q_1}S^2 \times ... \times T_{q_N}S^2 \to T_{q_1}S^2 \times ... \times T_{q_N}S^2, A_q(\omega) := R(q)\omega.$$

We can even notice that $R(q)$ defines a positive-definite bilinear form on this linear space, since

$$\omega^T R(q)\omega = \sum_{i,j=1}^{N} \omega_i^T \hat{q}_i^T M_{ij} \hat{q}_j \omega_j = \sum_{i,j=1}^{N} (\hat{q}_i\omega_i)^T M_{ij} (\hat{q}_j\omega_j) = v^T M v > 0.$$

The last inequality holds because M is the inertia matrix of the system and hence it defines a symmetric positive-definite bilinear form on $T_{q_1}S^2 \times ... \times T_{q_N}S^2$, see e.g. [16] [6]. This implies the map A_q is invertible and hence we are ready to

[6] It follows from the definition of the inertia tensor, i.e.

$$0 \leq \tilde{T}(q, \dot{q}) = \frac{1}{2}\sum_{i=1}^{N} \Big(\sum_{j\geq i} m_j\Big) L_i L_j \dot{q}_i^T \dot{q}_j := \frac{1}{2}\dot{q}^T M \dot{q}.$$

Moreover, in this situation it is even possible to explicitly find the Cholesky factorization of the matrix M with an iterative algorithm.

express the vector field in terms of the infinitesimal generator. We can rewrite the ODEs for the angular velocities as follows:

$$\dot{\omega} = A_q^{-1}\Big([g_1,...,g_N]^T\Big) = \begin{bmatrix} h_1(q,\omega) \\ ... \\ h_N(q,\omega) \end{bmatrix} = \begin{bmatrix} a_1(q,\omega)\times q_1 \\ ... \\ a_N(q,\omega)\times q_N \end{bmatrix},$$

where

$$g_i = g_i(q,\omega) = \sum_{\substack{j=1 \\ j\neq i}}^{N} M(q)_{ij}|\omega_j|^2\hat{q}_i q_j - \Big(\sum_{j=i}^{N} m_j\Big) g L_i \hat{q}_i e_3,\ i = 1,...,N$$

and $a_1,...,a_N : (TS^2)^N \to \mathbb{R}^3$ are N functions whose existence is guaranteed by the analysis done above. Indeed, we can set $a_i(q,\omega) := q_i \times h_i(q,\omega)$ and conclude that a mapping f from $(TS^2)^N$ to $(\mathfrak{se}(3))^N$ such that

$$\psi_*(f(q,\omega))|_{(q,\omega)} = F|_{(q,\omega)}$$

is the following one

$$f(q,\omega) = \begin{bmatrix} \omega_1 \\ q_1\times h_1 \\ ... \\ ... \\ \omega_N \\ q_N \times h_N \end{bmatrix} \in \mathfrak{se}(3)^N \simeq \mathbb{R}^{6N}.$$

We will not go into the Hamiltonian formulation of this problem; however, we remark that a similar approach works even in that situation. Indeed, following the derivation presented in [28], we see that for a mechanical system on $(S^2)^N$ the conjugate momentum writes

$$T^*_{q_1}S^2 \times ... T^*_{q_N}S^2 \ni \pi = (\pi_1,...,\pi_N),\ \text{where } \pi_i = -\hat{q}_i^2 \frac{\partial L}{\partial \omega_i}$$

and its components are still orthogonal to the respective base points $q_i \in S^2$. Moreover, Hamilton's equations take the form

$$\dot{q}_i = \frac{\partial H(q,\pi)}{\partial \pi_i} \times q_i,$$
$$\dot{\pi}_i = \frac{\partial H(q,\pi)}{\partial q_i} \times q_i + \frac{\partial H(q,\pi)}{\partial \pi_i} \times \pi_i,$$

which implies that setting

$$f(q,\pi) = \Big[\partial_{q_1}H(q,\pi),\quad \partial_{\pi_1}H(q,\pi),\quad \ldots,\quad \partial_{q_N}H(q,\pi),\quad \partial_{\pi_N}H(q,\pi)\Big]$$

we can represent even the Hamiltonian vector field of the $N-$fold 3D pendulum in terms of this group action.

Case $N = 2$

We have seen how it is possible to turn the equations of motion of a $N-$chain of pendulums into the Lie group integrators setting. Now we focus on the example with $N = 2$ pendulums. The equations of motion write

$$\dot{q}_1 = \hat{\omega}_1 q_1, \quad \dot{q}_2 = \hat{\omega}_2 q_2,$$

$$R(q) \begin{bmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \end{bmatrix} = \begin{bmatrix} (-m_2 L_1 L_2 |\omega_2|^2 \hat{q}_2 + (m_1 + m_2) g L_1 \hat{e}_3) q_1 \\ (-m_2 L_1 L_2 |\omega_1|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3) q_2 \end{bmatrix}, \tag{2.5.1}$$

where

$$R(q) = \begin{bmatrix} (m_1 + m_2) L_1^2 I_3 & m_2 L_1 L_2 \hat{q}_1^T \hat{q}_2 \\ m_2 L_1 L_2 \hat{q}_2^T \hat{q}_1 & m_2 L_2^2 I_3 \end{bmatrix}.$$

As presented above, the matrix $R(q)$ defines a linear invertible map of the space $T_{q_1} S^2 \times T_{q_2} S^2$ onto itself:

$$A_{(q_1,q_2)} : T_{q_1} S^2 \times T_{q_2} S^2 \to T_{q_1} S^2 \times T_{q_2} S^2, \, [\omega_1, \omega_2]^T \to R(q)[\omega_1, \omega_2]^T.$$

We can easily see that it is well defined since

$$R(q) \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} (m_1 + m_2) L_1^2 I_3 & m_2 L_1 L_2 \hat{q}_1^T \hat{q}_2 \\ m_2 L_1 L_2 \hat{q}_2^T \hat{q}_1 & m_2 L_2^2 I_3 \end{bmatrix} \begin{bmatrix} \hat{v}_1 q_1 \\ \hat{v}_2 q_2 \end{bmatrix} = \begin{bmatrix} \hat{r}_1 q_1 \\ \hat{r}_2 q_2 \end{bmatrix} \in (TS^2)^2$$

with

$$r_1(q, \omega) := (m_1 + m_2) L_1^2 v_1 + m_2 L_1 L_2 \hat{q}_2 \hat{v}_2 q_2,$$

$$r_2(q, \omega) := m_2 L_1 L_2 \hat{q}_1 \hat{v}_1 q_1 + m_2 L_2^2 v_2.$$

This map guarantees that if we rewrite the pair of equations for the angular velocities in (2.5.1) as

$$\dot{\omega} = R^{-1}(q) \begin{bmatrix} (-m_2 L_1 L_2 |\omega_2|^2 \hat{q}_2 + (m_1 + m_2) g L_1 \hat{e}_3) q_1 \\ (-m_2 L_1 L_2 |\omega_1|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3) q_2 \end{bmatrix} = R^{-1}(q) b =$$

$$= A^{-1}_{(q_1,q_2)}(b) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \in T_{q_1} S^2 \times T_{q_2} S^2,$$

then we are assured that there exists a pair of functions $a_1, a_2 : TS^2 \times TS^2 \to \mathbb{R}^3$ such that

$$\dot{\omega} = \begin{bmatrix} a_1(q, \omega) \times q_1 \\ a_2(q, \omega) \times q_2 \end{bmatrix} = \begin{bmatrix} h_1(q) \\ h_2(q) \end{bmatrix}.$$

Since we want $a_i \times q_i = h_i$, we just impose $a_i = q_i \times h_i$ and hence the whole vector field can be rewritten as

$$\begin{bmatrix} \dot{q}_1 \\ \dot{\omega}_1 \\ \dot{q}_2 \\ \dot{\omega}_2 \end{bmatrix} = \begin{bmatrix} \omega_1 \times q_1 \\ (q_1 \times h_1) \times q_1 \\ \omega_2 \times q_2 \\ (q_2 \times h_2) \times q_2 \end{bmatrix} = F|_{(q,\omega)},$$

with $h_i = h_i(q,\omega)$ and

$$\begin{bmatrix} h_1(q,\omega) \\ h_2(q,\omega) \end{bmatrix} = R^{-1}(q) \begin{bmatrix} (-m_2 L_1 L_2 |\omega_2|^2 \hat{q}_2 + (m_1 + m_2) g L_1 \hat{e}_3) q_1 \\ (-m_2 L_1 L_2 |\omega_1|^2 \hat{q}_1 + m_2 g L_2 \hat{e}_3) q_2 \end{bmatrix}.$$

Therefore, we can express the whole vector field in terms of the infinitesimal generator of the action of $SE(3) \times SE(3)$ as

$$\psi_*(f(q,\omega))|_{(q,\omega)} = F|_{(q,\omega)}$$

through the function

$$f : TS^2 \times TS^2 \to \mathfrak{se}(3) \times \mathfrak{se}(3) \simeq \mathbb{R}^{12}, \ (q,\omega) \to (\omega_1, q_1 \times h_1, \omega_2, q_2 \times h_2).$$

2.5.3 Numerical experiments

In this section, we present some numerical experiment for the $N-$chain of pendulums. We start by comparing the various Lie group integrators that we have tested (with the choice $N = 2$), and conclude by analyzing an implementation of variable step size. Lie group integrators allow to keep the evolution of the solution in the correct manifold, which is $TS^2 \times TS^2$ when $N = 2$. Hence, we briefly report two sets of numerical experiments. In the first one, we show the convergence rate of all the Lie group integrators tested on this model. In the second one, we check how they behave in terms of preserving the two following relations:

- $q_i(t)^T q_i(t) = 1$, i.e. $q_i(t) \in S^2$, $i = 1,2$,
- $q_i(t)^T \omega_i(t) = 0$, i.e. $\omega_i(t) \in T_{q_i(t)} S^2$, $i = 1,2$,

completing the analysis with a comparison with the classical Runge–Kutta 4 and with ODE45 of MATLAB. The Lie group integrators used to obtain the following experiments are Lie Euler, Lie Euler Heun, three versions of Runge–Kutta–Munthe–Kaas methods of order four and one of order three. The RKMK4 with two commutators mentioned in the plots, is the one presented in Section 2.2, while the other schemes can be found for example in [7].

Figure 2.5 presents the plots of the errors, in logarithmic scale, obtained considering as a reference solution the one given by the ODE45 method, with strict tolerance. Here, we used an exact expression for the $\mathrm{dexp}_\sigma^{-1}$ function. However, we could obtain the same results with a truncated version of this function, keeping a sufficiently high number of commutators, or after some clever manipulations of the commutators (as with RKMK4 with 2 commutators, see Section 2.2.2). The schemes show the right convergence rates, so we can move to the analysis of the time evolution on $TS^2 \times TS^2$.

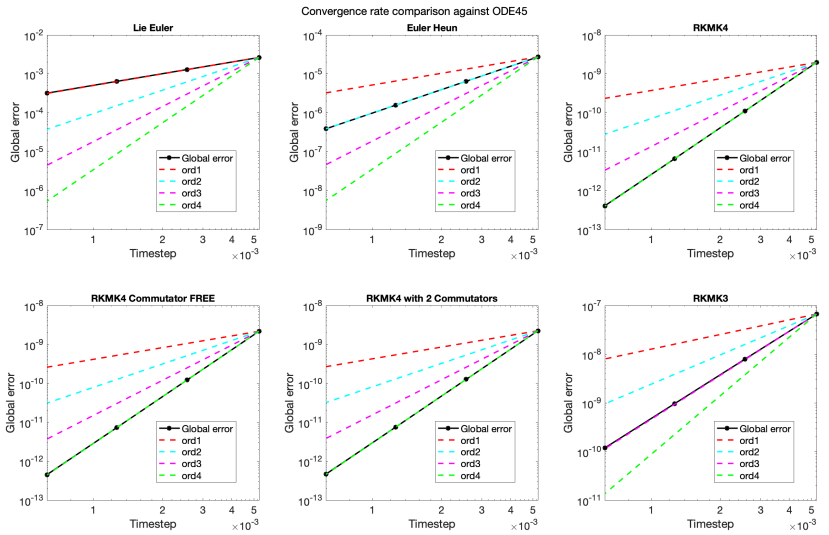


Figure 2.5: Convergence rate of the implemented Lie group integrators, based on global error considering as a reference solution the one of ODE45, with strict tolerance.

In Figure 2.6 we can see the comparison of the time evolution of the 2−norms of $q_1(t)$ and $q_2(t)$, for $0 \leq t \leq T = 5$. As highlighted above, unlike classical numerical integrators like the one implemented in ODE45 or the Runge–Kutta 4, the Lie group methods preserve the norm of the base components of the solutions, i.e. $|q_1(t)| = |q_2(t)| = 1 \ \forall t \in [0, T]$. Therefore, as expected, these integrators preserve the configuration manifold. However, to complete this analysis, we show the plots making a similar comparison but with the tangentiality conditions. Indeed, in Figure 2.7 we compare the time evolutions of the inner products $q_1(t)^T \omega_1(t)$ and $q_2(t)^T \omega_2(t)$ for $t \in [0,5]$, i.e. we see if these integrators preserve the geometry of the whole phase space $TS^2 \times TS^2$. As we can see, while for Lie group methods these inner products are of the order of 10^{-14} and 10^{-15}, the ones obtained with classical integrators

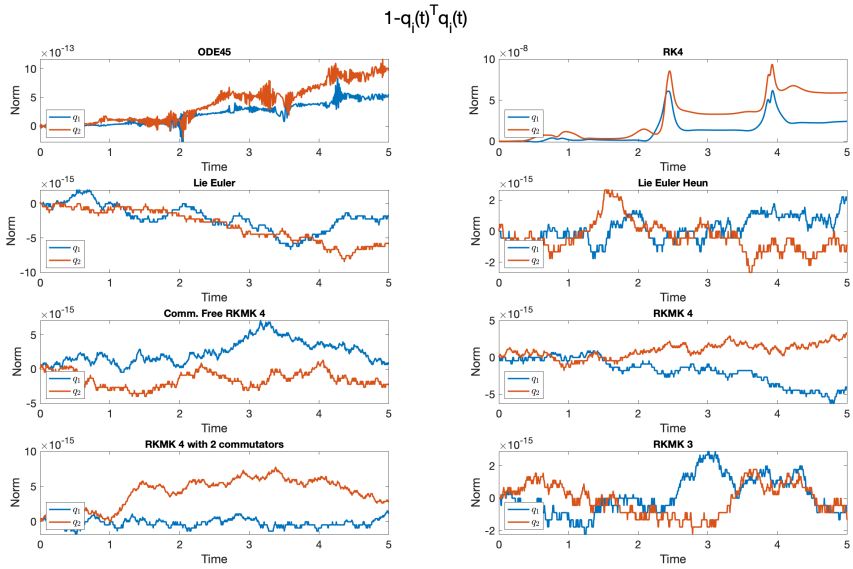


Figure 2.6: Visualization of the quantity $1 - q_i(t)^T q_i(t)$, $i = 1,2$, for time $t \in [0,5]$. These plots focus on the preservation of the geometry of S^2.

show that the tangentiality conditions are not preserved with the same accuracy.

We now move to some experiments on variable stepsize. In this last part we focus on the RKMK pair coming from Dormand–Prince method (DOPRI 5(4) [14]), which we denote with RKMK(5,4). The aim of the plots we show is to compare the same schemes, both with constant and variable stepsize. We start by setting a tolerance and solving the system with the RKMK(5,4) scheme. Using the same number of time steps, we solve it again with RKMK of order 5. These experiments show that, for some tolerance and some initial conditions, the step size's adaptivity improves the numerical approximation accuracy. Since we do not have an available analytical solution to quantify these two schemes' accuracy, we compare them with the solution obtained with a strict tolerance and ODE45. We compute such accuracy, at time $T = 3$, by means of the Euclidean norm of the ambient space $\mathbb{R}^{6N}$.

In Figure 2.8, we compare the performance of the constant and variable stepsize methods, where the structure of the initial condition is always the same, but what changes is the number of connected pendulums. The considered initial condition is $(q_i, \omega_i) = \left(\sqrt{2}/2, 0, \sqrt{2}/2, 0, 1, 0\right), \quad \forall\, i = 1, ..., N$, and all the masses and lengths are set to 1. From these experiments we can notice situations where the variable step size beats the constant one in terms of accuracy at the final

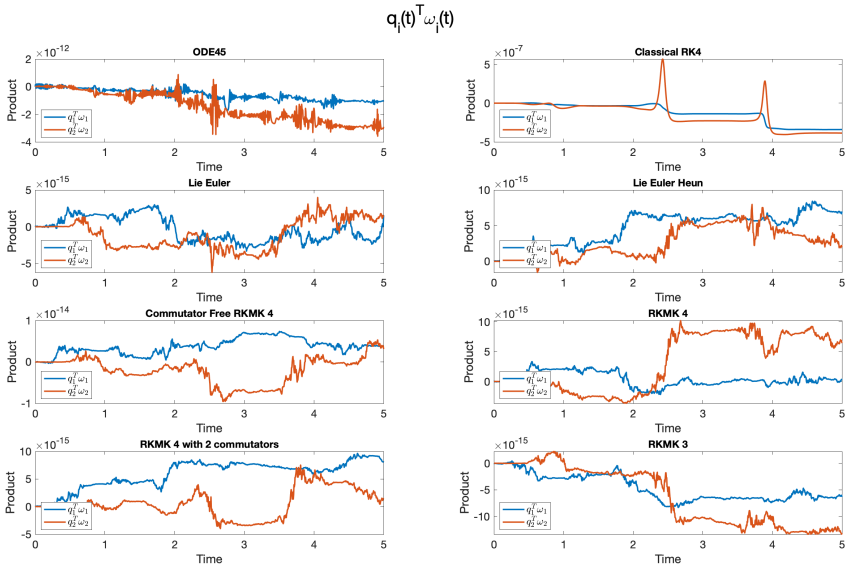


Figure 2.7: Visualization of the inner product $q_i(t)^T\omega_i(t)$, $i = 1,2$, for $t \in [0,5]$. These plots focus on the preservation of the geometry of $T_{q_i(t)}S^2$.

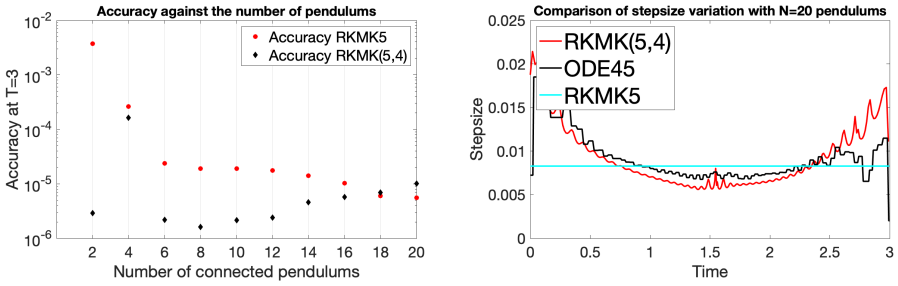
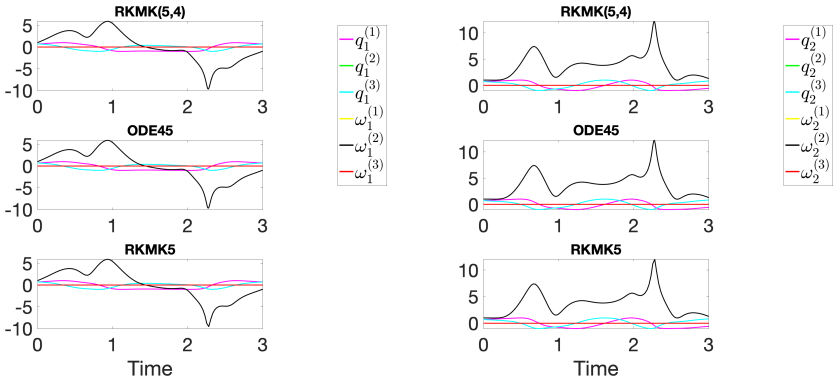


Figure 2.8: Comparison of accuracy at final time (on the left) and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 1$.

time, like the case $N = 2$ which we discuss in more detail afterwards.

The results presented in Figure 2.10 (left) do not aim to highlight any particular relation between how the number of pendulums increases or the regularity of the solution. Indeed, as we add more pendulums, we keep incrementing the total length of the chain since $\sum_{i=1}^{N} L_i = N$. Thus, here we do not have any appropriate limiting behaviour in the solution as $N \to +\infty$. The behaviour presented in that figure seems to highlight an improvement in accuracy for the RKMK5 method as N increases. However, this is biased by the fact that when we increase N, to achieve the fixed tolerance of 10^{-6} with RKMKK(5,4), we need more time steps in the discretization. Thus, this plot does not say that as N increases, the dynamics becomes more regular; it suggests that the number of required timesteps increases faster than the "degree of complexity" of the dynamics.



(a) $(q_1(t), \omega_1(t))$ **(b)** $(q_2(t), \omega_2(t))$

Figure 2.9: In these plots we represent the six components of the solution describing the dynamics of the first mass (on the left) and of the second mass (on the right), for the case $N = 2$. We compare the behaviour of the solution obtained with constant stepsize RKMK5, the variable stepsize RKMK(5,4) and ODE45.

For the case $N = 2$, we notice a relevant improvement passing to variable stepsize. In Figures 2.9 and 2.11 we can see that, for this choice of the parameters, the solution behaves smoothly in most of the time interval, but then there is a peak in the second component of the angular velocities of both the masses, at $t \approx 2.2$. We can observe this behaviour both in the plots of Figure 2.9, where we project the solution on the twelve components and even in Figure 2.11c. In the latter, we plot two of the vector field components, i.e. the second compo-

nents of the angular accelerations $\dot{\omega}_i(t)$, $i = 1,2$. They show an abrupt change in the vector field in correspondence to $t \approx 2.2$, where the step is considerably restricted. Thus, to summarize, the gain we see with variable stepsize when $N = 2$ is motivated by the unbalance in the length of the time intervals with no abrupt changes in the dynamics and those where they appear. Indeed, we see that apart from a neighbourhood of $t \approx 2.2$, the vector field does not change quickly. On the other hand, for the case $N = 20$, this is the case. Thus, the adaptivity of the stepsize does not bring relevant improvements in the latter situation.

The motivating application behind our choice of this mechanical system has been some intuitive relation with a beam model, as highlighted in the introduction of this work. However, for this limiting behaviour to make sense, we should fix the length of the entire chain of pendulums to some L (the length of the beam at rest) and then set the size of each pendulum to $L_i = L/N$. In this case, keeping the same tolerance of 10^{-6} for RKMK(5,4), we get the results presented in the following plot. We do not investigate more in details this approach, which might be relevant for further work, however we highlight that here the step adaptivity improves the results as we expected.

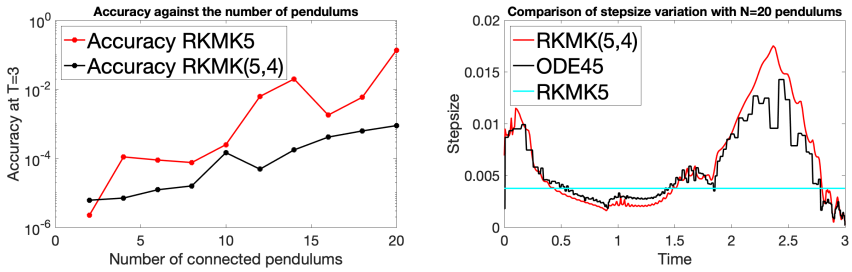
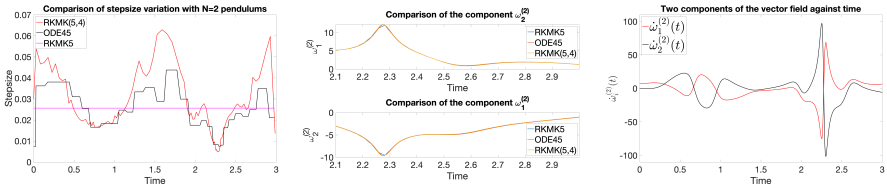


Figure 2.10: Comparison of accuracy at final time (on the left) and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 5/N$.

2.6 Dynamics of two quadrotors transporting a mass point

In this section we consider a multibody system made of two cooperating quadrotor unmanned aerial vehicles (UAV) connected to a point mass (suspended load) via rigid links. This model is described in [28, 50].

We consider an inertial frame whose third axis goes in the direction of gravity, but opposite orientation, and we denote with $y \in \mathbb{R}^3$ the mass point and with $y_1, y_2 \in \mathbb{R}^3$ the two quadrotors. We assume that the links between the two



(a) Step adaptation **(b)** Zoom at final times **(c)** Values of $\dot{\omega}_i^{(2)}(t)$

Figure 2.11: On the left, we compare the adaptation of the stepsize of RKMK(5,4) with the one of ODE45 and with the constant stepsize of RKMK5. In the center we plot the second component of the angular velocities $\omega_i^{(2)}$, $i = 1,2$, and we zoom in the last time interval $t \in [2.1,3]$ to see that the variable stepsize version of the method better reproduces the reference solution. On the right, we visualize the speed of variation of second component of the angular velocities.

quadrotors and the mass point are of a fixed length $L_1, L_2 \in \mathbb{R}^+$. The configuration variables of the system are: the position of the mass point in the inertial frame, $y \in \mathbb{R}^3$, the attitude matrices of the two quadrotors, $(R_1, R_2) \in (SO(3))^2$ and the directions of the links which connect the center of mass of each quadrotor respectively with the mass point, $(q_1, q_2) \in (S^2)^2$. The configuration manifold of the system is $Q = \mathbb{R}^3 \times (SO(3))^2 \times (S^2)^2$. In order to present the equations

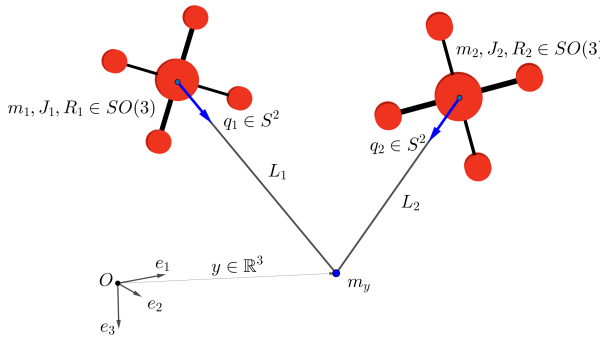


Figure 2.12: Two quadrotors connected to the mass point m_y via massless links of lengths L_i.

of motion of the system we start by identifying $TSO(3) \simeq SO(3) \times \mathfrak{so}(3)$ via left-trivialization. This choice allows us to write the kinematic equations of the system as

$$\dot{R}_i = R_i \hat{\Omega}_i, \quad \dot{q}_i = \hat{\omega}_i q_i \qquad i = 1,2, \tag{2.6.1}$$

where $\Omega_1, \Omega_2 \in \mathbb{R}^3$ represent the angular velocities of each quadrotor, respectively, and ω_1, ω_2 express the time derivatives of the orientations $q_1, q_2 \in S^2$, respectively, in terms of angular velocities, expressed with respect to the body-fixed frames. From these equations we define the trivialized Lagrangian

$$L(y, \dot{y}, R_1, \Omega_1, R_2, \Omega_2, q_1, \omega_1, q_2, \omega_2) : \mathbb{R}^6 \times \left(SO(3) \times \mathfrak{so}(3)\right)^2 \times (TS^2)^2 \to \mathbb{R},$$

as the difference of the total kinetic energy of the system and the total potential (gravitational) energy, $L = T - U$, with:

$$T = \frac{1}{2} m_y \|\dot{y}\|^2 + \frac{1}{2} \sum_{i=1}^{2} (m_i \|\dot{y} - L_i \hat{\omega}_i q_i\|^2 + \Omega_i^T J_i \Omega_i),$$

and

$$U = -m_y g e_3^T y - \sum_{i=1}^{2} m_i g e_3^T (y - L_i q_i),$$

where $J_1, J_2 \in \mathbb{R}^{3\times 3}$ are the inertia matrices of the two quadrotors and $m_1, m_2 \in \mathbb{R}^+$ are their respective total masses. In this system each of the two quadrotors generates a thrust force, which we denote with $u_i = -T_i R_i e_3 \in \mathbb{R}^3$, where T_i is the magnitude, while e_3 is the direction of this vector in the $i-$th body-fixed frame, $i = 1, 2$. The presence of these forces make it a non conservative system. Moreover, the rotors of the two quadrotors generate a moment vector, and we denote with $M_1, M_2 \in \mathbb{R}^3$ the cumulative moment vector of each of the two quadrotors. To derive the Euler–Lagrange equations, a possible approach is through Lagrange–d'Alambert's principle, as presented in [28]. We write them in matrix form as

$$A(z)\dot{z} = h(z) \tag{2.6.2}$$

where

$$z = [y, v, \Omega_1, \Omega_2, \omega_1, \omega_2]^T \in \mathbb{R}^{18},$$

$$A(z) = \begin{bmatrix} I_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & M_q & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & J_1 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & J_2 & 0_3 & 0_3 \\ 0_3 & -\frac{1}{L_1}\hat{q}_1 & 0_3 & 0_3 & I_3 & 0_3 \\ 0_3 & -\frac{1}{L_2}\hat{q}_2 & 0_3 & 0_3 & 0_3 & I_3 \end{bmatrix},$$

$$h(z) = \begin{bmatrix} h_1(z) \\ h_2(z) \\ h_3(z) \\ h_4(z) \\ h_5(z) \\ h_6(z) \end{bmatrix} = \begin{bmatrix} v \\ -\sum_{i=1}^{2} m_i L_i \|\omega_i\|^2 q_i + M_q g e_3 + \sum_{i=1}^{2} u_i^{\|} \\ -\Omega_1 \times J_1 \Omega_1 + M_1 \\ -\Omega_2 \times J_2 \Omega_2 + M_2 \\ -\frac{1}{L_1} g \hat{q}_1 e_3 - \frac{1}{m_1 L_1} q_1 \times u_1^{\perp} \\ -\frac{1}{L_2} g \hat{q}_2 e_3 - \frac{1}{m_2 L_2} q_2 \times u_2^{\perp} \end{bmatrix},$$

where $M_q = m_y I_3 + \sum_{i=1}^{2} m_i q_i q_i^T$, and $u_i^{\parallel}, u_i^{\perp}$ are respectively the orthogonal projection of u_i along q_i and to the plane $T_{q_i}S^2$, $i = 1,2$, i.e. $u_i^{\parallel} = q_i q_i^T u_i$, $u_i^{\perp} = (I - q_i q_i^T) u_i$. These equations, coupled with the kinematic equations in (2.6.1), describe the dynamics of a point

$$P = \left[y,\ v,\ R_1,\ \Omega_1,\ R_2,\ \Omega_2,\ q_1,\ \omega_1,\ q_2,\ \omega_2 \right] \in M = TQ.$$

Since the matrix $A(z)$ is invertible, we pass to the following set of equations

$$\dot{z} = A^{-1}(z)h(z) := \tilde{h}(z) := \bar{h}(P) = [\bar{h}_1(P), ..., \bar{h}_7(P)]^T. \tag{2.6.3}$$

2.6.1 Analysis via transitive group actions

We identify the phase space M with $M \simeq T\mathbb{R}^3 \times (TSO(3))^2 \times (TS^2)^2$. The group we consider is

$$\bar{G} = \mathbb{R}^6 \times (TSO(3))^2 \times (SE(3))^2,$$

where the groups are combined with a direct-product structure and $\mathbb{R}^6$ is the additive group. For a group element

$$g = ((a_1, a_2), ((B_1, b_1), (B_2, b_2)), ((C_1, c_1), (C_2, c_2))) \in \bar{G}$$

and a point $P \in M$ in the manifold, we consider the following left action

$$\begin{aligned} \psi_g(P) = [&y + a_1,\ v + a_2,\ B_1 R_1,\ \Omega_1 + b_1,\ B_2 R_2,\ \Omega_2 + b_2, \\ &C_1 q_1,\ C_1 \omega_1 + c_1 \times C_1 q_1,\ C_2 q_2,\ C_2 \omega_2 + c_2 \times C_2 q_2]. \end{aligned}$$

The well-definiteness and transitivity of this action come from standard arguments, see for example [42]. The infinitesimal generator associated to

$$\xi = \left[\xi_1,\ \xi_2,\ \eta_1,\ \eta_2,\ \eta_3,\ \eta_4,\ \mu_1,\ \mu_2,\ \mu_3,\ \mu_4 \right] \in \bar{\mathfrak{g}},$$

where $\bar{\mathfrak{g}} = T_e \bar{G}$, writes

$$\begin{aligned} \psi_*(\xi)|_P = [&\xi_1,\ \xi_2,\ \hat{\eta}_1 R_1,\ \eta_2,\ \hat{\eta}_3 R_2,\ \eta_4, \\ &\hat{\mu}_1 q_1,\ \hat{\mu}_1 \omega_1 + \hat{\mu}_2 q_1,\ \hat{\mu}_3 q_2,\ \hat{\mu}_3 \omega_2 + \hat{\mu}_4 q_2]. \end{aligned}$$

We can now focus on the construction of the function $f : M \to \bar{\mathfrak{g}}$ such that $\psi_*(f(P))|_P = F|_P$, where

$$\begin{aligned} F|_P = [&\bar{h}_1(P),\ \bar{h}_2(P),\ R_1 \hat{\Omega}_1,\ \bar{h}_3(P),\ R_2 \hat{\Omega}_2, \\ &\bar{h}_4(P),\ \hat{\omega}_1 q_1,\ \bar{h}_5(P),\ \hat{\omega}_2 q_2,\ \bar{h}_6(P)] \in T_P M \end{aligned}$$

is the vector field obtained combining the equations (2.6.1) and (2.6.3). We have

$$f(P) = [\bar{h}_1(P),\ \bar{h}_2(P),\ R_1\Omega_1,\ \bar{h}_3(P),\ R_2\Omega_2,\ \bar{h}_4(P),\\ \omega_1,\ q_1 \times \bar{h}_5(P),\ \omega_2,\ q_2 \times \bar{h}_6(P)] \in \tilde{\mathfrak{g}}.$$

We have obtained the local representation of the vector field $F \in \mathfrak{X}(M)$ in terms of the infinitesimal generator of the transitive group action ψ, hence we can solve for one time step Δt the IVP

$$\begin{cases} \dot{\sigma}(t) = \mathrm{dexp}^{-1}_{\sigma(t)}\Big(f\big(\psi(\exp(\sigma(t)), P(t))\big)\Big) \\ \sigma(0) = 0 \in \tilde{\mathfrak{g}} \end{cases}$$

and then update the solution $P(t + \Delta t) = \psi(\exp(\sigma(\Delta t)), P(t))$.

The above construction is completely independent of the control functions $\{u_i^{\parallel}, u_i^{\perp}, M_i\}_{i=1,2}$ and hence it is compatible with any choice of these parameters.

2.6.2 Numerical experiments

We tested Lie group numerical integrators for a load transportation problem presented in [50]. The control inputs $\{u_i^{\parallel}, u_i^{\perp}, M_i\}_{i=1,2}$ are constructed such that the point mass asymptotically follows a given desired trajectory $y_d \in \mathbb{R}^3$, given by a smooth function of time, and the quadrotors maintain a prescribed formation relative to the point mass. In particular, the parallel components $u_i^{\parallel}$ are designed such that the payload follows the desired trajectory y_d (load transportation problem), while the normal components $u_i^{\perp}$ are designed such that q_i converge to desired directions q_{id} (tracking problem in S_2). Finally, M_i are designed to control the attitude of the quadrotors.

In this experiment we focus on a simplified dynamics model, i.e. we neglect the construction of the controllers M_i for the attitude dynamics of the quadrotors. However, the full dynamics model can also be easily integrated, once the expressions for the attitude controllers are available.

In Figure 2.13 we show the convergence rate of four different RKMK methods compared with the reference solution obtained with ODE45 in MATLAB.

In Figures 2.14-2.18 we show results in the tracking of a parabolic trajectory, obtained by integrating the system (2.6.2) with a RKMK method of order 4.

2.7 Summary and outlook

In this paper we have considered Lie group integrators with a particular focus on problems from mechanics. In mathematical terms this means that the Lie

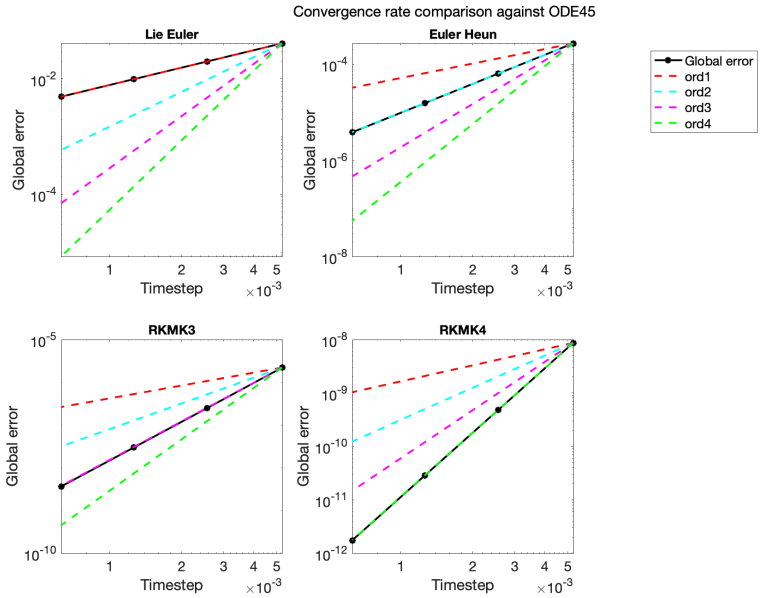


Figure 2.13: Convergence rate of the numerical schemes compared with ODE45

groups and manifolds of particular interest are $SO(n),\ n=2,3$, $SE(n),\ n=2,3$ as well as the manifolds S^2 and TS^2. The abstract formulations by e.g. Crouch and Grossman [11], Munthe-Kaas [40] and Celledoni et al. [6] have often been demonstrated on small toy problems in the literature, such as the free rigid body or the heavy top systems. But in papers like [4], hybrid versions of Lie group integrators have been applied to more complex beam and multi-body problems. The present paper is attempting to move in the direction of more relevant examples without causing the numerical solution to depend on how the manifold is embedded in an ambient space, or the choice of local coordinates.

It will be the subject of future work to explore more examples and to aim for a more systematic approach to applying Lie group integrators to mechanical problems. In particular, it is of interest to the authors to consider models of beams, that could be seen as a generalisation of the N-fold pendulum discussed here.

Acknowledgments This work was supported by Marie Sklodowska-Curie [860124].

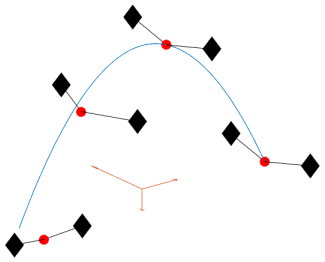


Figure 2.14: Snapshots at $0 \leq t \leq 5$.

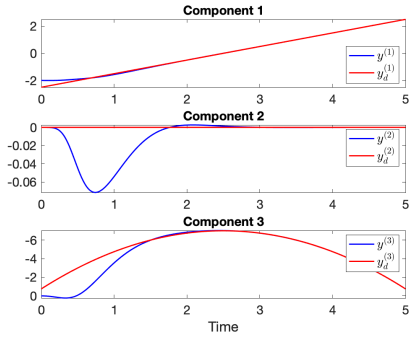


Figure 2.15: Components of the load position (in blue) and the desired trajectory (in red) as a function time.

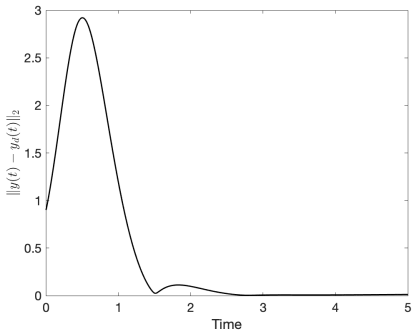


Figure 2.16: Deviation of the load position from the target trajectory.

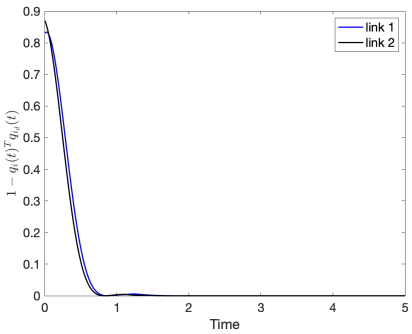


Figure 2.17: Direction error of the links.

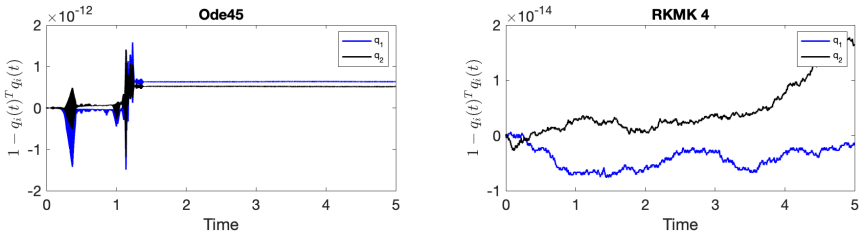


Figure 2.18: Preservation of the norms of $q_1, q_2 \in S^2$.

Bibliography

[1] M. Arnold and O. Brüls. Convergence of the generalized-α scheme for constrained mechanical systems. *Multibody Syst. Dyn.*, 18(2):185–202, 2007. 32

[2] M. Arnold, O. Brüls, and A. Cardona. Error analysis of generalized-α Lie group time integration methods for constrained mechanical systems. *Numer. Math.*, 129(1):149–179, 2015. 32

[3] G. Bogfjellmo and H. Marthinsen. High-order symplectic partitioned Lie group methods. *Foundations of Computational Mathematics*, pages 1–38, 2015. 41

[4] O. Bruls and A. Cardona. On the Use of Lie Group Time Integrators in Multibody dynamics. *J. Computational Nonlinear Dynamics*, 5(3):031002, 2010. 32, 44, 45, 65

[5] F. Casas and B. Owren. Cost Efficient Lie Group Integrators in the RKMK Class. *BIT Numerical Mathematics*, 43(4):723–742, 2003. 37

[6] E. Celledoni, A. Marthinsen, and B. Owren. Commutator-free Lie group methods. *Future Generation Computer Systems*, 19:341–352, 2003. 32, 33, 38, 65

[7] E. Celledoni, H. Marthinsen, and B. Owren. An introduction to Lie group integrators—basics, new developments and applications. *J. Comput. Phys.*, 257(part B):1040–1061, 2014. 32, 55

[8] E. Celledoni and B. Owren. Lie group methods for rigid body dynamics and time integration on manifolds. *Comput. Methods Appl. Mech. Engrg.*, 192(3-4):421–438, 2003. 38

[9] J. Ćesić, V. Jukov, I. Petrovic, and D. Kulić. Full body human motion estimation on lie groups using 3D marker position measurements. *In Proceedings of the IEEE-RAS International Conference on HumanoidRobotics, Cancun, Mexico, 15–17 November 2016*, 2016. 33

[10] S.H. Christiansen, H.Z. Munthe-Kaas, and B. Owren. Topics in structure-preserving discretization. *Acta Numerica*, 20:1–119, 2011. 32

[11] P. E. Crouch and R. Grossman. Numerical integration of ordinary differential equations on manifolds. *J. Nonlinear Sci.*, 3:1–33, 1993. 32, 33, 34, 38, 65

[12] C. Curry and B. Owren. Variable step size commutator free Lie group integrators. *Numer. Algorithms*, 82(4):1359–1376, 2019. 44, 47, 48

[13] F. Diele, L. Lopez, and R. Peluso. The Cayley transform in the numerical solution of unitary differential systems. *Adv. Comput. Math.*, 8(4):317–334, 1998. 32, 39

[14] J. R Dormand and P. J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980. 57, 80

[15] K. Engø and S. Faltinsen. Numerical integration of Lie–Poisson systems while preserving coadjoint orbits and energy. *SIAM J. Numer. Anal.*, 39(1):128–145, 2001. 41, 44

[16] H. Goldstein, C.P. Poole, and J. Safko. *Classical Mechanics*. Pearson, 2013. 52

[17] V. Guillemin and S. Sternberg. The moment map and collective motion. *Ann. Physics*, 127:220–253, 1980. 43

[18] E. Hairer, Ch. Lubich, and G. Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2010. Structure-preserving algorithms for ordinary differential equations, Reprint of the second (2006) edition. 32

[19] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Verlag, Second revised edition, 1993. 45

[20] J. Hall and M. Leok. Lie group spectral variational integrators. *Found. Comput. Math.*, 17(1):199–257, 2017. 32

[21] F. Hausdorff. Die symbolische Exponentialformel in der Gruppentheorie. *Leipziger Ber.*, 58:19–48, 1906. 37

[22] H. M. Hilber, T. J. R. Hughes, and R. L. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5(3):283–292, 1977. cited By 1683. 32

[23] D. Holm. *Geometric Mechanics: Part II: Rotating, Translating and Rolling*. World Scientific Publishing Company, 2008. 44, 50

[24] D. Holm, J. Marsden, and T. Ratiu. The Euler–poincaré equations and semidirect products with applications to continuum theories. *Adv. in Math.*, 137:1–81, 1998. 41

[25] S. Holzinger and J. Gerstmayr. Time integration of rigid bodies modelled with three rotation parameters. *Multibody Sys Dyn*, 1–34, 2021. 33

[26] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna. Lie-group methods. *Acta Numerica*, 9:215–365, 2000. 32

[27] T. Lee, M. Leok, and N. H. McClamroch. Lie group variational integrators for the full body problem. *Comput. Methods Appl. Mech. Engrg.*, 196(29-30):2907–2924, 2007. 32, 33

[28] T. Lee, M. Leok, and N. H. McClamroch. *Global formulations of Lagrangian and Hamiltonian dynamics on manifolds*. Interaction of Mechanics and Mathematics. Springer, Cham, 2018. A geometric approach to modeling and analysis. 32, 48, 51, 53, 60, 62

[29] T. Leitz and S. Leyendecker. Galerkin Lie-group variational integrators based on unit quaternion interpolation. *Comput. Methods Appl. Mech. Engrg.*, 338:333–361, 2018. 32

[30] N. E. Leonard and J. E. Marsden. Stability and drift of underwater vehicle dynamics: Mechanical systems with rigid motion symmetry. *Physica D*, 105(1–3):130–162, 1997. 44

[31] D. Lewis and J. C. Simo. Conserving algorithms for the dynamics of Hamiltonian systems of Lie groups. *J. Nonlinear Sci.*, 4:253–299, 1994. 32, 41, 43

[32] A. Lundervold and H. Z. Munthe-Kaas. On algebraic structures of numerical integration on vector spaces and manifolds. In *Faà di Bruno Hopf algebras, Dyson-Schwinger equations, and Lie-Butcher series*, volume 21 of *IRMA Lect. Math. Theor. Phys.*, pages 219–263. Eur. Math. Soc., Zürich, 2015. 35

[33] J. E. Marsden, T. Ratiu, and A. Weinstein. Semi-direct products and reduction in mechanics. *Transactions of the American Mathematical Society*, 281:147–77, 1884. 41

[34] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer-Verlag, 1994. 35

[35] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Number 17 in Texts in Applied Mathematics. Springer-Verlag, second edition, 1999. 41

[36] R. H. Merson. An operational method for the study of integration processes. In *Proc. Symp. Data Processing*, 1957. 45

[37] J. Moser and A. P. Veselov. Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Comm. Math. Phys.*, 139(2):217–243, 1991. 32

[38] A. Müller. Coordinate mappings for rigid body motions. *ASME Journal of Computational and Nonlinear Dynamics*, 12, 2017. 39

[39] H. Munthe-Kaas. Runge–Kutta methods on Lie groups. *BIT*, 38(1):92–111, 1998. 33

[40] H. Munthe-Kaas. High order Runge–Kutta methods on manifolds. *Appl. Numer. Math.*, 29:115–127, 1999. 32, 33, 34, 37, 65, 97

[41] H. Munthe-Kaas and B. Owren. Computations in a free Lie algebra. *Phil. Trans. Royal Soc. A*, 357:957–981, 1999. 37

[42] P. J. Olver. *Applications of Lie groups to differential equations*, volume 107. Springer Science & Business Media, 2000. 50, 63

[43] B. Owren. Order conditions for commutator-free Lie group methods. *J. Phys. A*, 39(19):5585–5599, 2006. 47

[44] B. Owren. Lie group integrators. In *Discrete mechanics, geometric integration and Lie-Butcher series*, volume 267 of *Springer Proc. Math. Stat.*, pages 29–69. Springer, Cham, 2018. 32

[45] B. Owren and A. Marthinsen. Integration methods based on canonical coordinates of the second kind. *Numer. Math.*, 87(4):763–790, 2001. 32, 39

[46] J. Park and W. Chung. Geometric integration on Euclidean group with application to articulated multibody systems. *J. CAM*, 21, 2005. 33

[47] T. Ratiu. Euler-Poisson equations on Lie algebras and the n-dimensional heavy rigid body. *Proc. Nat. Acad. Sci. USA*, pages 1327–1328, 1981. 43

[48] A. Saccon. Midpoint rule for variational integrators on Lie groups. *Internat. J. Numer. Methods Engrg.*, 78(11):1345–1364, 2009. 43

[49] J. C. Simo and L. Vu-Quoc. On the dynamics of finite-strain rods undergoing large motions — a geometrically exact approach. *Comput. Methods Appl. Mech. Engrg.*, 66:125–161, 1988. 32, 33

[50] Lee T., K. Sreenath, and V. Kumar. Geometric control of cooperating multiple quadrotor uavs with a suspended payload. *52nd IEEE Conference on Decision and Control*, pages 5510–5515, 2013. 60, 64

[51] A. P. Veselov. Integrable systems with discrete time, and difference operators. *Funktsional. Anal. i Prilozhen.*, 22(2):1–13, 96, 1988. 32

[52] A. M. Vinogradov and B. Kupershmidt. The structure of Hamiltonian mechanics. *Funktsional. Anal. i Prilozhen.*, 32:177–243, 1977. 43

[53] F. W. Warner. *Foundations of Differentiable Manifolds and Lie Groups*. GTM 94. Springer-Verlag, 1983. 34

[54] V. Wieloch and M. Arnold. BDF integrators for constrained mechanical systems on Lie groups. *J. CAM*, 387:112517, 2019, 2019. 32

Dynamics of the N-fold Pendulum in the framework of Lie Group Integrators

Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari and Brynjulf Owren

European Consortium for Mathematics in Industry. Cham: Springer International Publishing, 2021. p. 297-304

Dynamics of the N-fold Pendulum in the framework of Lie Group Integrators

Abstract. Since their introduction, Lie group integrators have become a method of choice in many application areas. Various formulations of these integrators exist, and in this work we focus on Runge–Kutta–Munthe–Kaas methods. First, we briefly introduce this class of integrators, considering some of the practical aspects of their implementation, such as adaptive time stepping. We then present some mathematical background that allows us to apply them to some families of Lagrangian mechanical systems. We conclude with an application to a nontrivial mechanical system: the N-fold 3D pendulum.

3.1 Introduction

Lie group integrators are used to simulate problems whose solution evolves on a manifold. Many approaches to Lie group integrators can be found in the literature, with several applications for mechanical systems (see, e.g. [2], [8], [3]).

The present work is motivated by applications in modelling and simulation of slender structures like beams, and the example considered here is a chain of pendulums. The dynamics of this mechanical system is described in terms of a Lie group G acting transitively on the phase space $\mathcal{M}$. This setting is used to build also a numerical integrator.

In Section 3.2 we give a brief overview of the Runge–Kutta–Munthe–Kaas (RKMK) methods with particular focus on the variable step size methods, which we use later in subsection 3.4.2 for the numerical experiments.

In Section 3.3 we introduce some necessary mathematical background that allows us to apply RKMK methods to the system of interest. In particular, we focus on a condition that guarantees the homogeneity of the tangent bundle TQ of a manifold Q. We then consider Cartesian products of homogeneous manifolds.

In Section 3.4 we reframe the ODE system of the chain of N connected $3D$ pendulums in the geometric framework presented in Section 3.3. We write the equations of motion and represent them in terms of the infinitesimal generator of the transitive action. The final part shows some numerical experiments where the constant and variable step size methods are compared.

3.2 RKMK methods with variable step size

The underlying idea of RKMK methods is to express a vector field $F \in \mathfrak{X}(\mathcal{M})$ as $F|_m = \psi_*(f(m))|_m$, where ψ_* is the infinitesimal generator of ψ, a transitive

action on $\mathcal{M}$, and $f : \mathcal{M} \to \mathfrak{g}$. This allows us to transform the problem from the manifold $\mathcal{M}$ to the Lie algebra $\mathfrak{g}$, on which we can perform a time step integration. We then map the result back to $\mathcal{M}$, and repeat this up to the final integration time. More explicitly, let h_n be the size of the $n-$th time step, we then update $y_n \in \mathcal{M}$ to y_{n+1} by

$$\begin{cases} \sigma(0) = 0 \in \mathfrak{g}, \\ \dot{\sigma}(t) = \mathrm{dexp}^{-1}_{\sigma(t)} \circ f \circ \psi(\exp(\sigma(t)), y_n) \in T_{\sigma(t)}\mathfrak{g}, \\ y_{n+1} = \psi(\exp(\sigma_1), y_n) \in \mathcal{M}, \end{cases} \tag{3.2.1}$$

where $\sigma_1 \approx \sigma(h_n) \in \mathfrak{g}$ is computed with a Runge-Kutta method.

One approach for varying the step size is based on embedded Runge–Kutta pairs for vector spaces. This approach consists of a principal method of order p, used to propagate the numerical solution, together with some auxiliary method, of order $\tilde{p} < p$, that is only used to obtain an estimate of the local error. This local error estimate is in turn used to derive a step size adjustment formula that attempts to keep the local error estimate approximately equal to some user-defined tolerance tol in every step. Both methods are applied to solve the ODE for $\sigma(t)$ in (3.2.1), yielding two approximations σ_1 and $\tilde{\sigma}_1$ respectively, using the same step size h_n. Now, some distance measure between σ_1 and $\tilde{\sigma}_1$ provides an estimate e_{n+1} for the size of the local truncation error. Thus, $e_{n+1} = Ch_{n+1}^{\tilde{p}+1} + \mathcal{O}(h^{\tilde{p}+2})$. Aiming at $e_{n+1} \approx \mathrm{tol}$ in every step, one may use a formula of the type

$$h_{n+1} = \theta \left(\frac{\mathrm{tol}}{e_{n+1}} \right)^{\frac{1}{\tilde{p}+1}} h_n \tag{3.2.2}$$

where θ is typically chosen between 0.8 and 0.9. If $e_n > \mathrm{tol}$, the step is rejected. Hence, we can redo the step with the step size obtained by the same formula.

3.3 Mathematical background

This section introduces the mathematical background that allows us to study many mechanical systems in the framework of Lie group integrators and Lie group actions. In particular, we provide some results that we use to study the model of a chain of N 3D-pendulums presented in the last section.

3.3.1 The tangent bundle of some homogeneous manifolds is homogeneous

For Lagrangian mechanical systems, the phase space is usually the tangent bundle TQ of some configuration manifold Q. In [1] the authors present a

setting in which the homogeneity of Q implies that of TQ. We now briefly review and reframe it in the notation used throughout the paper.

Consider a smooth homogeneous $n-$dimensional manifold Q. This means that Q is endowed with a transitive G-group action $\Lambda : G \times Q \to Q$, i.e., for any pair $q_1, q_2 \in Q$ there is $g \in G$ such that $\Lambda(g, q_1) = q_2$. Assume that for each $q \in Q$, the map $\Lambda_q : G \to Q$ defined as $\Lambda_q(g) := \Lambda(g, q)$, is a submersion at $e \in G$. When these hypotheses hold, it can be shown that TQ is a homogeneous manifold as well, and an explicit transitive action can be obtained from Λ. Let Λ_* be the infinitesimal generator of the group action Λ, and denote with $\bar{\xi}(q) := \Lambda_*(\xi)(q) \in T_qQ$ the differential at the identity element $e \in G$ of Λ_q, evaluated at $\xi \in \mathfrak{g}$. We then introduce $\Lambda_g : Q \to Q$, $q \mapsto \Lambda(g, q)$ and call $T_{\bar{q}}\Lambda_g$ its tangent lift at $\bar{q} \in Q$.

Consider the manifold $\bar{G} := G \ltimes \mathfrak{g}$, equipped with the semi-direct product Lie group structure (see, e.g., [5]). We can introduce a transitive group action on TQ as follows:

$$\varphi : \bar{G} \times TQ \to TQ, \ ((g, \xi), (q, v)) \mapsto \Big(\Lambda(g, q), \bar{\xi}(\Lambda(g, q)) + T_q\Lambda_g(v)\Big).$$

By direct computation and basic properties of Lie groups (see, e.g., [6]), it can be seen that the action φ is well defined. Since the action Λ is transitive on Q and Λ_q is assumed to be a submersion at $e \in G$, we have that

$$\forall v' \in T_{q'}Q \ \exists \xi \in \mathfrak{g} \text{ s.t. } \Lambda_*(\xi)(q') = \bar{\xi}(\Lambda(g, q)) = v' - T_q\Lambda_g(v).$$

Thus, we conclude that $\mathcal{M} = TQ$ is a homogeneous manifold.

In the application treated in the next section, we are interested in the case in which $Q = S^2 \subset \mathbb{R}^3$, i.e., the unit sphere. In this setting, a transitive group action Λ is given by

$$\Lambda : SO(3) \times S^2 \to S^2, \ (R, q) \mapsto Rq,$$

$$T_qS^2 \ni \Lambda_*(\xi)(q) = \bar{\xi}(q) = \xi \times q, \quad T_q\Lambda_R(v) = Rv \in T_{Rq}S^2.$$

Therefore, in this case we recover the restriction to $TS^2 \subset \mathbb{R}^6 \simeq \mathfrak{se}(3)$ of the Adjoint action of $\bar{G} = SE(3) = SO(3) \ltimes \mathbb{R}^3 \simeq SO(3) \ltimes \mathfrak{so}(3)$ (see, e.g., [7])

$$\varphi((R, r), (q, v)) = (Rq, Rv + r \times Rq) = {}^{1}(Rq, Rv + \hat{r}Rq) \tag{3.3.1}$$

which hence becomes a particular case of a more general framework.

[1] Here $\hat{r} = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}$, where $r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$.

3.3.2 The Cartesian product of homogeneous manifolds is homogeneous

Consider a family of homogeneous manifolds $\mathcal{M}_1, ..., \mathcal{M}_n$. Call $(G_i, \odot_i)$ the Lie group acting transitively on the associated smooth manifold $\mathcal{M}_i$, and φ_i such a transitive action. Let $\mathfrak{g}_i$ be the Lie algebra of G_i, $i = 1, ..., n$, and

$$\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times ... \times \mathcal{M}_n, \quad G = G_1 \times G_2 \times ... \times G_n.$$

The manifold G can be naturally equipped with a Lie group structure given by the direct product. More precisely, for a pair of elements $G \ni g_i = (g_i^1, ..., g_i^n)$, $i = 1, 2$, we can define their product $g_1 \cdot g_2 := (g_1^1 \odot_1 g_2^1, ..., g_1^n \odot_n g_2^n) \in G$. We can similarly define componentwise the exponential map.

This construction ensures that the manifold $\mathcal{M}$ is homogeneous too, and G acts transitively on it. That is, let

$$g = (g^1, ..., g^n) \in G, \quad m = (m^1, ..., m^n) \in \mathcal{M},$$

then

$$\varphi : G \times \mathcal{M} \to \mathcal{M}, \quad \varphi(g, m) := (\varphi_1(g^1, m^1), ..., \varphi_n(g^n, m^n)).$$

We now restrict to the specific case $\mathcal{M}_i = TS^2$ for $i = 1, ..., n$. Since TS^2 is a homogeneous manifold with transitive action φ defined as in equation (3.3.1), we can write the transitive group action

$$\psi : (SE(3))^n \times (TS^2)^n \to (TS^2)^n,$$

$$\psi((g^1, ..., g^n), (m^1, ..., m^n)) = (\varphi(g^1, m^1), ..., \varphi(g^n, m^n)),$$

where $g^i := (R_i, r_i) \in SE(3)$, $m^i = (q_i, v_i) \in TS^2$.

3.4 The N-fold 3D pendulum

We now apply the geometric setting from section 3.3 to the specific problem of a chain of N connected 3D pendulums, whose dynamics evolves on $(TS^2)^N$.

3.4.1 Equations of motion

Let us consider a chain of N pendulums subject to constant gravity g. The system is modeled by N rigid, massless links serially connected by spherical joints, with the first link connected to a fixed point placed at the origin of the ambient space $\mathbb{R}^3$, as in figure 3.1. We neglect friction and interactions among the pendulums.

The modeling part comes from [9] and we omit details. We denote by $q_i \in S^2$ the configuration vector of the $i-$th mass, m_i, of the chain. Following

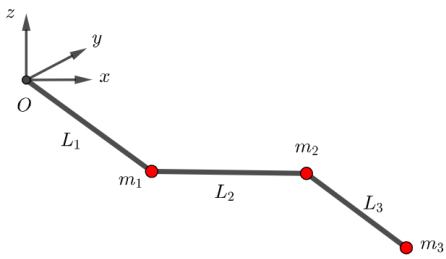


Figure 3.1: Chain of 3 connected pendulums at a fixed time instant.

[**?**], we express the Euler–Lagrange equations for our system in terms of the configuration variables $(q_1,\dots,q_N) \in (S^2)^N \subset \mathbb{R}^{3N}$, and their angular velocities $(\omega_1,...,\omega_N) \in T_{q_1}S^2 \times ... \times T_{q_N}S^2 \subset \mathbb{R}^{3N}$, defined be the following kinematic equations:

$$\dot{q}_i = \omega_i \times q_i, \quad i = 1,\dots,N. \tag{3.4.1}$$

The Euler–Lagrange equations of the system can be written as

$$R(q)\dot{\omega} = \left[\sum_{\substack{j=1 \\ j\neq i}}^{N} M_{ij}|\omega_j|^2 \hat{q}_i q_j - \Big(\sum_{j=i}^{N} m_j\Big) g L_i \hat{q}_i e_3 \right]_{i=1,...,N} = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} \in \mathbb{R}^{3N}, \tag{3.4.2}$$

where $R(q) \in \mathbb{R}^{3N\times 3N}$ is a symmetric block matrix defined as

$$R(q)_{ii} = \Big(\sum_{j=i}^{N} m_j\Big) L_i^2 I_3 \in \mathbb{R}^{3\times 3},$$

$$R(q)_{ij} = \Big(\sum_{k=j}^{N} m_k\Big) L_i L_j \hat{q}_i^T \hat{q}_j \in \mathbb{R}^{3\times 3} = R(q)_{ji}^T,\ i < j,$$

and

$$M_{ij} = \Big(\sum_{k=\max\{i,j\}}^{N} m_k\Big) L_i L_j I_3 \in \mathbb{R}^{3\times 3}.$$

Equations (3.4.1)-(3.4.2) define the dynamics of the N-fold pendulum, and hence a vector field $F \in \mathfrak{X}((TS^2)^N)$. We now find a function $f : (TS^2)^N \to \mathfrak{se}(3)^N$ such that

$$\psi(f(m))|_m = F|_m,\ \forall m \in (TS^2)^N,$$

where ψ is defined as in subsection 3.3.2.

Since $R(q)$ defines a linear invertible map (see [2])

$$A_q : T_{q_1}S^2 \times ... \times T_{q_N}S^2 \to T_{q_1}S^2 \times ... \times T_{q_N}S^2, \quad A_q(\omega) := R(q)\omega,$$

we can rewrite the ODEs for the angular velocities as follows:

$$\dot{\omega} = A_q^{-1}\left(\begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}\right) = \begin{bmatrix} h_1(q,\omega) \\ \vdots \\ h_N(q,\omega) \end{bmatrix} = \begin{bmatrix} a_1(q,\omega) \times q_1 \\ \vdots \\ a_N(q,\omega) \times q_N \end{bmatrix}. \tag{3.4.3}$$

In equation (3.4.3) the r_is are defined as in (3.4.2), and $a_1, ..., a_N : (TS^2)^N \to \mathbb{R}^3$ can be defined as $a_i(q,\omega) := q_i \times h_i(q,\omega)$. Thus, the map f is given by

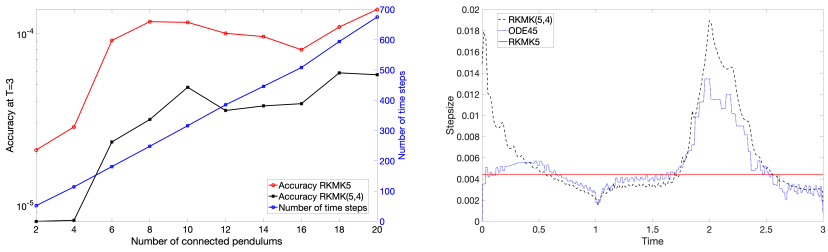
$$f(q,\omega) = \begin{bmatrix} \omega_1 \\ q_1 \times h_1(q,\omega) \\ \vdots \\ \omega_N \\ q_N \times h_N(q,\omega) \end{bmatrix} \in \mathfrak{se}(3)^N \simeq \mathbb{R}^{6N}.$$

3.4.2 Numerical experiments

In this section we show a numerical experiment with the N-fold 3D pendulum, in which we compare the performance of constant and variable step size methods. We do not show results on the preservation of the geometry (up to machine accuracy), since this is given by construction. We consider the RKMK pair coming from Dormand–Prince method (DOPRI 5(4) [4], which we denote by RKMK(5,4)). We set a tolerance of 10^{-6} and solve the system with the RKMK(5,4) scheme. Fixing the number of time steps required by RKMK(5,4), we repeat the experiment with RKMK of order 5 (denoted by RKMK5). The comparison occurs at the final time $T = 3$ using the Euclidean norm of the ambient space $\mathbb{R}^{6N}$. The quality of the approximation is measured against a reference solution obtained with ODE45 from MATLAB with a strict tolerance.

The motivating application behind the choice of this mechanical system has been some intuitive relation with flexible slender structures like beams. For this limiting behaviour to make sense, we first fix the length of the entire chain of pendulums to some L, then we set the size of each pendulum to $L_i = L/N$ and initialize $(q_i, \omega_i) = \big(1,0,0,0,0,0\big)$, $\forall i = 1, ..., N$. As we can see in figure 3.2a, the results of our experiments show that number of time steps that RKMK(5,4) requires to reach the desired accuracy increases with N, and this can be read in terms of an augmentation of the dynamics' complexity. For this reason, as highlighted in figure 3.2, distributing these time steps uniformly in the time interval $[0, T]$ becomes an inefficient approach, and hence a variable step size method gives better performance.

We further design a slightly different experiment to compare the computational time of the constant and variable stepsize RKMK methods. First, we



(a) Accuracy against the number of pendulums **(b)** Comparison of step sizes with 20 pendulums

Figure 3.2: Comparisons of variable versus constant stepsize for the N-fold 3D pendulum

fix the tolerance $tol = 10^{-6}$ for RKMK(5,4) and compute its distance from the reference solution with ODE45. Then, we aim to replicate this error with RKMK5, increasing the number of performed time steps. We report in Table 3.1 the results of the experiment. Because of the more efficient distribution of the time steps, we notice smaller values with RKMK(5,4) for the more involved systems.

Pendulums	2	4	6	8	10	12	14	16	18	20
RKMK5	0.12	0.42	1.04	2.24	3.80	6.74	9.09	12.71	18.51	27.67
RKMK(5,4)	0.16	0.38	0.91	1.59	2.83	4.51	6.93	9.71	13.68	18.81
Ratio	0.75	1.11	1.14	1.41	1.34	1.49	1.31	1.31	1.35	1.47

Table 3.1: Elapsed times (in seconds) obtained with RKMK5 (second row) and with RKMK(5,4) (third row) for systems having different number of pendulums (first row). In the last row we report the ratio between the RKMK5 and the RKMK(5,4) runtimes. These are obtained with the `tic–toc` command of MATLAB.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124.

Bibliography

[1] R. W. Brockett and H. J. Sussmann. Tangent bundles of homogeneous spaces are homogeneous spaces. *Proceedings of the American Mathematical Society*, 35(2):550–551, 1972. 76, 97

[2] E. Celledoni, E. Çokaj, A. Leone, D. Murari, and B. Owren. Lie group integrators for mechanical systems. *International Journal of Computer Mathematics*, 99(1):58–88, 2022. 75, 79, 85, 97, 100

[3] Elena Celledoni, Håkon Marthinsen, and Brynjulf Owren. An introduction to lie group integrators–basics, new developments and applications. *Journal of Computational Physics*, 257:1040–1061, 2014. 75

[4] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980. 57, 80

[5] K. Engø. Partitioned Runge–Kutta methods in Lie-group setting. *BIT Numerical Mathematics*, 43:21–39, 2003. 77

[6] B.C. Hall. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Graduate Texts in Mathematics. Springer, 2015. 77

[7] D. D. Holm, T. Schmah, and C. Stoica. *Geometric mechanics and symmetry: from finite to infinite dimensions*, volume 12. Oxford University Press, 2009. 77

[8] Arieh Iserles, Hans Z Munthe-Kaas, Syvert P Nørsett, and Antonella Zanna. Lie-group methods. *Acta numerica*, 9:215–365, 2000. 75

[9] T. Lee, M. Leok, and N. H. McClamroch. *Global Formulations of Lagrangian and Hamiltonian Dynamics on Manifolds: A Geometric Approach to Modeling and Analysis*. Interactions of Mechanics and Mathematics. Springer, 2018. 78

Learning Hamiltonians of constrained mechanical systems

Elena Celledoni, Andrea Leone, Davide Murari and Brynjulf Owren

Journal of Computational and Applied Mathematics, 2023, 417: 114608

Learning Hamiltonians of constrained mechanical systems

Abstract. Recently, there has been an increasing interest in modelling and computation of physical systems with neural networks. Hamiltonian systems are an elegant and compact formalism in classical mechanics, where the dynamics is fully determined by one scalar function, the Hamiltonian. The solution trajectories are often constrained to evolve on a submanifold of a linear vector space. In this work, we propose new approaches for the accurate approximation of the Hamiltonian function of constrained mechanical systems given sample data information of their solutions. We focus on the importance of the preservation of the constraints in the learning strategy by using both explicit Lie group integrators and other classical schemes.

4.1 Introduction

Neural networks have been proven to be effective in learning patterns from data in many different contexts. Recently there has been an increasing interest in applying neural networks to learn physical models from data, for example models of classical mechanics. For Hamiltonian systems, multiple approaches have been proposed to approximate the energy function, see, e.g., [9], [14], [26], [13], [23]. Building on these results, we propose an improved learning procedure. Our main contribution is an approach to learn the Hamiltonian for systems defined on the cotangent bundle $T^*\mathcal{Q}$ of some manifold $\mathcal{Q}$ embedded in a vector space. Under the assumption that $T^*\mathcal{Q}$ is homogeneous, we show how to do that while preserving the geometry during the learning phase. In this paper, by preservation of the geometry we mean the accurate conservation of the constraints rather than of other geometric features such as symplecticity, energy or other first integrals of the system.

As in [13], we express the dynamics of constrained systems by embedding the problem in a vector space of larger dimension, but in our approach we do not make use of Lagrange multipliers. With the aim of understanding the importance of the geometry in this approximation problem, we compare learning procedures based on numerical integrators that preserve the phase space of the system with others that do not. We restrict to homogeneous spaces where Lie group methods can preserve the geometry up to machine accuracy (see, e.g., [8]). For example, multi-body lumped mass systems fall naturally in this setting [20, Chapter 2]. This restriction still includes systems with the configuration manifold that is a Lie group, as in some problems of rigid body and rod dynamics, but we will not consider these applications here. The experiments show that there are specific problems where approximating the Hamiltonian

using a Lie group method can be relevant. Surprisingly, in many other settings classical Runge–Kutta integrators produce comparable results.

The main focus of the present paper is to learn an approximation of a Hamiltonian system where the training data are given as a set of trajectory segments. To do so, one could learn the dynamics either by approximating the Hamiltonian vector field or the Hamiltonian function as done in our work. Another relevant difference in the learning framework consists of considering in the training procedure either one time step of the flow map (see, e.g., [14]) or a sequence of successive time steps as proposed in [9]. In the latter work it is shown with experimental evidence that taking into account temporal dependencies improves performance. We follow the second strategy when dealing with unconstrained systems, whereas we test both of them with our approach to constrained systems.

In principle, the Hamiltonian can be any differentiable function. However, for mechanical systems, it is often made by the sum of (quadratic) kinetic energy and a potential energy, [25], [15], [21]. Following [26], we make the ansatz that the kinetic energy is characterized by a symmetric and positive definite matrix, and hence we aim to estimate it.

We conclude this Section with a more precise definition of the problem of interest. In the second Section, we introduce the Hamiltonian formalism for both unconstrained and constrained systems. In the third Section, we focus on unconstrained systems, presenting the general learning procedure that will be extended to constrained systems in the fourth Section. We also discuss how additional known information about the dynamical system can be included in the network training procedure. The experimental results show that physics-based regularization could be helpful to improve the extrapolation capability of the network and its stability in the presence of noise. In the last Section, we formalize the problem of learning a constrained Hamiltonian mechanical system and discuss the importance of the geometry for this class of problems. Finally, we complete this Section with numerical experiments in the `PyTorch` framework, showing how the predicted Hamiltonian depends on some training parameters and on the presence of noise. The numerical implementations are available in the GitHub repository associated to the paper[1].

4.1.1 Description of the problem

Suppose to be given a set of N sampled trajectories coming from a Hamiltonian system defined on a submanifold $\mathcal{M} = T^*\mathcal{Q}$ of $\mathbb{R}^{2n}$, where $T^*\mathcal{Q}$ is the cotangent bundle of the configuration manifold $\mathcal{Q}$ (see [19][Chapter 11] for more details). Moreover, assume that each of these trajectories contains M

[1] `https://github.com/davidemurari/learningConstrainedHamiltonians`

equispaced (in time) points. In other words, suppose that

$$\{(x_i, \bar{y}_i^2, ..., \bar{y}_i^M)\}_{i=1,...,N},\ \bar{y}_i^j = \Phi_{X_H}^{(j-1)\Delta t}(x_i) \tag{4.1.1}$$

as a training set, where $\Phi_{X_H}^t$ is the time t-flow of the exact, unknown Hamiltonian system. In practice, we never have access to the exact trajectories but to either a noisy version of them or a numerical approximation.

The approach we use aims to approximate the vector field $X_H \in \mathfrak{X}(\mathcal{M})$ that governs the dynamics, where by $\mathfrak{X}(\mathcal{M})$ we denote the collection of all smooth vector fields on $\mathcal{M}$.. However, we know that such a vector field is Hamiltonian, i.e. there exists a scalar function $H : \mathcal{M} \to \mathbb{R}$ which, together with the geometry given by $\mathcal{M}$, characterizes the dynamics completely. For this reason, we do not need to directly approximate X_H, but just H and then eventually recover X_H.

The problem under consideration can be described as an inverse problem, since we want to infer the function H from trajectory data of the corresponding dynamical system rather than from samples of the function H itself. This description of the problem motivates how we measure the accuracy of our approximation, denoted by a parametric model H_Θ. Indeed, the target is not to approximate the trajectories of the given Hamiltonian system with some neural network, but to approximate the Hamiltonian. Thus the quality of the approximation can be computed in at least two ways. First, one can compare some measured trajectories with those obtained from the approximation. More precisely, we randomly generate $\tilde{N}$ initial conditions $z_i \in \mathcal{M}$, their $\tilde{M}$ time updates, and compute

$$\mathcal{E}_1\left(\left\{u_i^j\right\}_{i=1,...,\tilde{N}}^{j=1,...,\tilde{M}}, \left\{v_i^j\right\}_{i=1,...,\tilde{N}}^{j=1,...,\tilde{M}}\right) = \frac{1}{\tilde{N}\tilde{M}} \sum_{j=1}^{\tilde{M}} \sum_{i=1}^{\tilde{N}} \left\| u_i^j - v_i^j \right\|^2, \tag{4.1.2}$$

where $\|\cdot\|$ is the Euclidean norm of $\mathbb{R}^{2n}$, $u_i^1 = z_i$, $v_i^1 = z_i$, $u_i^{j+1} = \Psi_{X_H}^h(u_i^j)$ and $v_i^{j+1} = \Psi_{X_{H_\Theta}}^h(v_i^j)$ for a numerical integrator Ψ^h of choice. One can randomly generate these initial conditions for academic examples where the true Hamiltonian is actually known. In this case, $\tilde{N}$ and $\tilde{M}$ can be specified arbitrarily, usually with $\tilde{N}$ less the the number of training trajectories N. On the other hand, in more realistic applications one has to work with the initial conditions for which the related trajectory segments are known. In this case $\tilde{N}$ and $\tilde{M}$ are constrained by the available data, in particular the number of total trajectories is split into N for training and $\tilde{N}$ for test. In our experiments, we adopted the `SciPy` implementation of the Dormand-Prince pair of order (5,4) with a strict tolerance. In fact, following the PyTorch implementation of the mean squared error, $\mathcal{E}_1$ is actually divided by $2n$. Alternatively, as introduced in [11], one can compare pointwise values of the approximated and the true Hamiltonian, when

known. This gives

$$\mathcal{E}_2(H, H_\Theta) = \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} \left| H(z_i) - H_\Theta(z_i) - \frac{1}{\tilde{N}} \sum_{l=1}^{\tilde{N}} \left(H(z_l) - H_\Theta(z_l) \right) \right|, \tag{4.1.3}$$

where $\mathcal{E}_2$ handles the fact that Hamiltonians differing, on $\mathcal{M}$, by a constant generate the same vector field. Indeed, $\mathcal{E}_2(H, H + c) = 0$.

4.2 Hamiltonian mechanical systems

In this work, we focus on Hamiltonian mechanical systems based on a configuration manifold $\mathcal{Q} \subseteq \mathbb{R}^n$. We now introduce some basic elements of the theory of unconstrained Hamiltonian dynamics on $\mathbb{R}^{2n}$, which corresponds to the case $\mathcal{Q} = \mathbb{R}^n$. Then we extend this formulation to constrained systems on $T^*\mathcal{Q} \subset \mathbb{R}^{2n}$.

The Hamiltonian formalism gives a particular class of conservative vector fields which, in contrast to the Lagrangian one, can always be expressed with a system of first-order ordinary differential equations. For the unconstrained case, the equations are of the form $\dot{x}(t) = \mathbb{J}\nabla H(x(t)) := X_H(x(t))$ where $x(t) = [q(t), p(t)] \in \mathbb{R}^{2n}$ comprises the configuration variables and their conjugate momenta. Here, $H : \mathbb{R}^{2n} \to \mathbb{R}$ is a smooth function called the Hamiltonian of the system, and $\mathbb{J} \in \mathbb{R}^{2n \times 2n}$ is the symplectic matrix.

In this work, we focus on Hamiltonian systems whose energy function is of the form

$$H(q, p) = \frac{1}{2} p^T M^{-1}(q) p + V(q)$$

where $M(q)$ is the mass matrix of the system, possibly depending on the configuration $q \in \mathbb{R}^n$, and $V(q)$ is the potential energy of the system. This is not a too restrictive assumption since it still includes a quite broad family of systems. For unconstrained systems, we will further restrict to the case where M is a constant matrix and the Hamiltonian is separable. This assumption allows to implement symplectic numerical integration without needing implicit updates. On the other hand, in the constrained setting we aim at preserving the geometry of the numerical flow map rather than other properties such as symplecticity. As a consequence, we can work with variable mass matrices still using explicit numerical integrators as in the unconstrained case.

We now briefly formalize how to extend this formulation to Hamiltonian systems that are holonomically constrained on some configuration manifold $\mathcal{Q} = \{q \in \mathbb{R}^n : g(q) = 0\}$ embedded in $\mathbb{R}^n$ (for a more detailed derivation of this formalism we refer to [20, Chapter 8]). Many mechanical systems relevant for applications are characterized by the presence of some constraints that are coupled to the ODE defining the dynamics. One way to model this kind of

problems is based on Lagrange multipliers, which lead to differential algebraic equations (DAEs). There has been some work in the direction of extending the Hamiltonian neural network's framework to constrained systems (see, e.g., [13] in which this strategy of introducing Lagrange multipliers is applied).

In this manuscript, we want to present an alternative approach based on the assumption that the constrained manifold $\mathcal{Q}$ is embedded in some linear space $\mathbb{R}^n$. This is actually not a restriction, since Whitney's embedding theorem always guarantees the existence of such an ambient space (see, e.g., [19, Chapter 6]). More explicitly, because of this embedding property, constrained multibody systems can be modelled by means of some projection operator and the vector field is written in such a way that it directly respects the constraints, without the addition of algebraic equations.

Furthermore, we assume that the components $g_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, ..., m$, are functionally independent on the zero level set, so that the Hamiltonian is defined on the $(2n-2m)$ dimensional cotangent bundle $\mathcal{M} = T^*\mathcal{Q}$. Working with elements of the tangent space at q, $T_q\mathcal{Q}$, as vectors in $\mathbb{R}^n$, we introduce a linear operator that defines the orthogonal projection of an arbitrary vector $v \in \mathbb{R}^n$ onto $T_q\mathcal{Q}$, i.e.

$$\forall q \in \mathcal{Q}, \text{ we set } P(q) : \mathbb{R}^n \to T_q\mathcal{Q},\ v \mapsto P(q)v.$$

$P(q)^T$ can be seen as a map sending vectors of $\mathbb{R}^n$ into covectors in $T_q^*\mathcal{Q}$. If $g(q)$ is differentiable, assuming $G(q)$ is the Jacobian matrix of $g(q)$, we have $T_q\mathcal{Q} = \operatorname{Ker} G(q)$, and so $P(q) = I_n - G(q)\left(G(q)^T G(q))\right)^{-1} G(q)^T$, where $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix. This projection map allows us to define Hamilton's equations as follows

$$\begin{cases} \dot{q} = P(q)\partial_p H(q,p) \\ \dot{p} = -P(q)^T \partial_q H(q,p) + W(q,p)\partial_p H(q,p), \end{cases} \tag{4.2.1}$$

where

$$W(q,p) = P(q)^T \Lambda(q,p)^T P(q) + \Lambda(q,p)P(q) - P(q)^T \Lambda(q,p)^T,$$
$$\text{with} \quad \Lambda(q,p) = \frac{\partial P(q)^T p}{\partial q}.$$

It is important to remark that since $T^*\mathcal{Q} \subset \mathbb{R}^{2n}$, we can work with the coordinates of the ambient space in the subsequent development. We notice that when $\mathcal{Q} = \mathbb{R}^n$, we can set $P(q) = I$ and recover the unconstrained formulation. These equations of motion can be derived by the standard Hamilton's variational principle on the phase space or by the Legendre transform applied to the Euler-Lagrange equations. However, due to the geometry of the system, the

variations need to be constrained to the right spaces and this is done with the projection map $P(q)$. We will focus on the case $Q = S^2 \times ... \times S^2 = (S^2)^k$ in Section 4.4.2, where the mass matrix $M(q)$ and equation (4.2.1) takes a structured form, with S^2 the unit sphere in $\mathbb{R}^3$.

4.3 Learning unconstrained systems

As in [9], we base the training on a recurrent approach, that is graphically described in Figure 4.1.

As mentioned in Subsection 4.1.1, we work with numerically generated training trajectories that we denote by

$$\{(x_i, y_i^2, ..., y_i^M)\}_{i=1,...,N}.$$

We limit the treatment of noisy training data to Subsection 4.3.2. To obtain an approximation of the Hamiltonian H, we define a parametric model H_Θ and look for a Θ so that the trajectories generated by H_Θ resemble the given ones. H_Θ in principle can be any parametric function depending on the parameters Θ. In our approach, Θ will collect a factor of the mass matrix and the weights of a neural network, as specified in equation (4.3.3). We use some numerical one-step method $\Psi^{\Delta t}_{X_{H_\Theta}}$ to generate the trajectories

$$\hat{y}_i^j(\Theta) := \Psi^{\Delta t}_{X_{H_\Theta}}(\hat{y}_i^{j-1}(\Theta)), \quad \hat{y}_i^1(\Theta) := x_i, \quad j = 2, \dots, M,\ i = 1, \dots, N. \tag{4.3.1}$$

For unconstrained problems we use symplectic numerical integrators, since they can take an explict form and their adoption in the training procedure allows to have a target modified Hamiltonian to approximate (see, e.g., [27]). We then optimize a loss function measuring the distance between the given trajectories y_i^j and the generated ones $\hat{y}_i^j$, defined as

$$\mathcal{L}(\Theta) := \frac{1}{2n}\frac{1}{NM}\sum_{i=1}^{N}\mathcal{L}_i(\Theta) = \frac{1}{2n}\frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\|\hat{y}_i^j(\Theta) - y_i^j\|^2, \tag{4.3.2}$$

where $\|\cdot\|$ is the Euclidean metric of $\mathbb{R}^{2n}$. This is implemented with the PyTorch `MSELoss` loss function. Such a training procedure resembles the one of Recurrent Neural Networks (RNNs), introduced in [24], as shown for the forward pass of a single training trajectory in Figure 4.1. Indeed, the weight sharing principle of RNNs is reproduced by the time steps in the numerical integrator which are all based on the same approximation of the Hamiltonian, and hence on the same weights Θ. Finally, in Algorithm 4.1 we report one training epoch for a batch of data points.

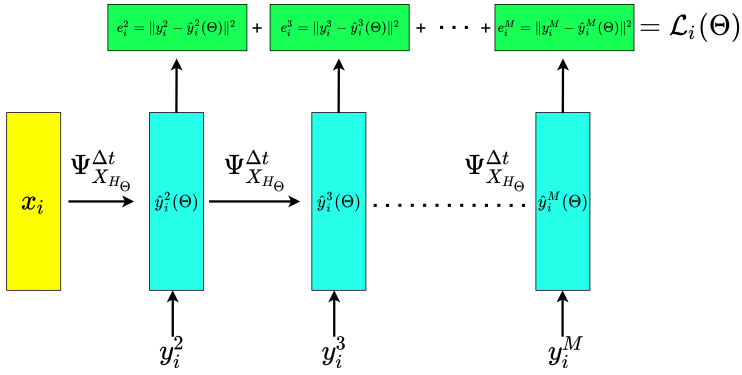


Figure 4.1: Forward pass of an input training trajectory $(x_i, y_i^2, ..., y_i^M)$. The picture highlights the resemblance to an unrolled version of a Recurrent Neural Network. The network outputs $(\hat{y}_i^2, \dots, \hat{y}_i^M)$.

Algorithm 4.1 One epoch of the recurrent approximation of the Hamiltonian.

Choose a numerical integrator (s stages)
$\hat{N} \leftarrow$ `batch size`, Loss $\leftarrow 0$
for $i = 1, \dots, \hat{N}$ **do**
  $\hat{y}_i^1 \leftarrow x_i$
  for $j = 1, \dots, M$ **do**
    $\hat{y}_i^{j,[1]} \leftarrow \hat{y}_i^j$
    for $k = 1, \dots, s-1$ **do**
      Compute current value of Hamiltonian $H_\Theta(\hat{y}_i^{j,[k]})$
      Compute $\nabla H_\Theta(\hat{y}_i^{j,[k]})$ ▷ With automatic differentiation
      Compute stage $\hat{y}_i^{j,[k+1]}$
    end for
    Compute $\hat{y}_i^{j+1}$
    Increase Loss following equation (4.3.2)
  end for
end for
Optimize Loss

4.3.1 Architecture of the network

In this work, the role of the neural network is to model the Hamiltonian, i.e. a scalar function defined on the phase space $\mathbb{R}^{2n}$. Thus, the starting and arrival spaces are fixed. For unconstrained systems we assume that

$$H(q,p) = \frac{1}{2} p^T M^{-1} p + V(q) = K(p) + V(q)$$

is separable. Here, the kinetic energy is a quadratic form defined by the symmetric positive definite matrix M^{-1}. It can hence be modelled through a learnable matrix A, $K(p) \approx K_A(p)$, by replacing M^{-1} or M with $A^T A$ during the learning procedure. This modelling choice improves extrapolation properties since it allows to learn local (on a compact set) information that is valid on a larger domain, i.e. the mass matrix. In Section 4.4 we extend this reasoning to some configuration dependent mass matrices, where $M(q)$ is modelled through a constant symmetric and positive definite matrix. Recalling that $A^T A$ can even be singular or close to singular, one can promote the positive definiteness of the modelled matrix adding a positive definite perturbation matrix to $A^T A$. Notice that, in principle, the imposition of the positive (semi)definiteness of the matrix defining the kinetic energy is not necessary, but it allows to get more interpretable results. Indeed, it is known that the kinetic energy should define a metric on $\mathbb{R}^n$ and the assumption we are making guarantees such a property. For constrained systems we proceed in a similar way, as shown in equation (4.4.3). For the potential energy, a possible modelling strategy is to work with standard feedforward neural networks, and hence to define

$$V(q) \approx V_\theta(q) = f_{\theta_m} \circ ... \circ f_{\theta_1}(q),$$
$$\theta_i = (W_i, b_i) \in \mathbb{R}^{n_i \times n_{i-1}} \times \mathbb{R}^{n_i},\ \theta := [\theta_1, ..., \theta_m],$$
$$f_{\theta_i}(u) := \Sigma(W_i u + b_i),\ \mathbb{R}^n \ni z \mapsto \Sigma(z) = [\sigma(z_1), ..., \sigma(z_n)] \in \mathbb{R}^n,$$

for example with $\sigma(x) = \tanh(x)$. In particular applications, where some additional information is known about the system, one can impose more structure on the architecture modelling $V(q)$. For example, in the case of odd potential or rotationally symmetric potential, one can define respectively an odd neural network V_θ or a rotationally equivariant one (see, e.g., [4]). Therefore, we have that

$$\Theta = [A, \theta], \quad H(q,p) \approx H_\Theta(q,p) = K_A(p) + V_\theta(q). \tag{4.3.3}$$

We remark that the Hamiltonian does not need to be approximated by a neural network, and hence in a compositional way. Many other parametrizations are possible. For example, starting from the sparse identification of dynamical systems approach presented in [3], in [12] it is proposed to parametrize the Hamiltonian with a dictionary of functions, for example polynomials and trigonometric functions. In our work, however, we opt for standard feedforward neural networks as the modelling assumption.

We now provide further details on the extrapolation capabilities of this network model. The learning procedure presented above is based on extracting temporal information coming from a set of trajectories belonging to a compact subset $\Omega \subset \mathbb{R}^{2n}$. In general, there is no reason why the Hamiltonian should be accurate outside of this set. To be more precise, denoting by $T > 0$ the largest time at which we know the trajectories, we have that

1. given enough samples in set Ω, distributed in order to capture the behaviour of the dynamical system, the prediction of the network is expected to be accurate in $\Omega_{[0,T]} := \{\Phi^t_{X_H}(x) : t \in [0,T],\, x \in \Omega\}$, i.e. for any $z_0 \in \Omega_{[0,T]}$ and any $\bar{t} > 0$ such that $\Phi^t(z_0) \in \Omega_{[0,T]}$ for all $t \in [0, \bar{t}]$,

2. outside $\Omega_{[0,T]}$ one cannot guarantee that the prediction will be accurate.

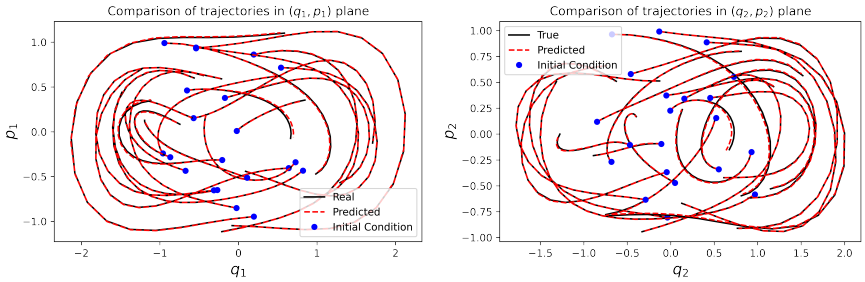
If we think of classical regression problems or even classification ones, it seems reasonable not to have information about the approximated quantity outside the sampled area. In those cases, with generalization we mean being sufficiently accurate close to the training points but still inside the sampled domain. However, here we know that the inferred function $H(q,p)$ has physical meaning and properties, so we might incorporate global known information about it to extend the applicability of the predictions.

This discussion supports the architectural choice for the kinetic energy suggested before (as in [26]). Indeed, supposing the Hamiltonian is separable, we know that the variable p appears in the energy function only via the quadratic form $\frac{1}{2}p^T M^{-1} p$. Thus, our modelling assumption allows us to approximate the mass matrix M just from a set of trajectories, hence capturing the dependency of H on the variable p also outside $\Omega_{[0,T]}$. Other possible improvements can be obtained when some symmetry structure is known for the Hamiltonian. On a similar direction, in Subsection 4.3.2, we add some regularization based on other prior physical knowledge.

We present in Figure 4.2 the comparison between ten learned trajectories and the corresponding exact ones of the Hamiltonian system $X_H \in \mathfrak{X}(\mathbb{R}^4)$ with Hamiltonian

$$H(q,p) = \frac{1}{2}\begin{bmatrix} p_1 & p_2 \end{bmatrix}^T \begin{bmatrix} 5 & -1 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \frac{q_1^4 + q_2^4}{4} + \frac{q_1^2 + q_2^2}{2}. \tag{4.3.4}$$

The training procedure of the network is based on 900 trajectories, sampled uniformly in 6 time instants, on the interval $0 \le t \le 0.3$. We remark that the training initial conditions are carefully chosen so that their associated trajectory segments well-capture the dynamics of interest. Figure 4.2 collects test trajectories corresponding to the time interval $[0,1]$. Since we are interested in approximating the Hamiltonian and not directly the trajectories, we are not constrained to evaluate the quality of the approximation with the same time integrator as the one used for training. In fact, these test trajectories have been generated with an embedded Runge–Kutta pair of order (5,4), with same relative and absolute accuracies for both the real and learned systems. Experimentally, it is clear that the qualitative behaviour of the Hamiltonian is well captured, as we can see from Figure 4.2. To quantify the agreement of the prediction with the true Hamiltonian we report the $\mathcal{E}_1$ metric, as defined in (4.1.2), that is $6.59 \cdot 10^{-5}$. Furthermore, the training loss is $4.62 \cdot 10^{-7}$.



(a) Projection on (q_1, p_1) **(b)** Projection on (q_2, p_2)

Figure 4.2: Comparison of real and predicted test trajectories for the Hamiltonian (4.3.4). In this case, for the potential energy we used a feedforward network with 3 hidden layers having respectively 100, 50 and 50 neurons and tanh as activation function. The training integrator is Störmer-Verlet, with $M = 6$ and final time $T = 0.3$ and we use the Adam optimizer. The test trajectories, at $\tilde{M} = 20$ uniformly distributed points in the time interval $[0, 1]$, are obtained with ODE(5,4). These trajectories correspond to $\tilde{N} = 100$ initial conditions on which the network has not been trained.

4.3.2 Robustness to noise and regularization

In real world applications, data is contaminated by noise which usually comes from the measurement process. Thus, we need to test the robustness of the learning framework to the presence of noise in the training trajectories. To do so, we synthetically generate the trajectories as before, and then add random normal noise to all the points except the initial condition (for an averaging strategy that allows to deal even with perturbed initial conditions, see, e.g., [9]). By construction, the network necessarily learns a Hamiltonian function, that is expected to generate trajectories close to the noisy ones. Since the training does not rely on clean trajectories, it is reasonable not to expect neither a loss value which is as small as in the absence of noise, nor a too accurate approximation of the Hamiltonian and the trajectories. Nevertheless, we aim for a learned Hamiltonian with level sets close to the exact ones, hence giving trajectories that resemble the true ones. One way of improving the quality of the neural networks proposed here, is to make use of a priori known physical properties of the dynamical system. We use an approach based on soft constraints which means that we take the known physical properties into account by adding a regularization term in the cost function. An example of such a property could be one or more known conserved quantities, so called first integrals. Hamiltonian systems always have at least one first integral, namely the Hamiltonian function itself, but there might be additional independent ones. Enforcing the first

integrals to be preserved or nearly preserved seems to be a reasonable strategy for obtaining improved qualitative behaviour of the resulting approximation as shown in the following example.

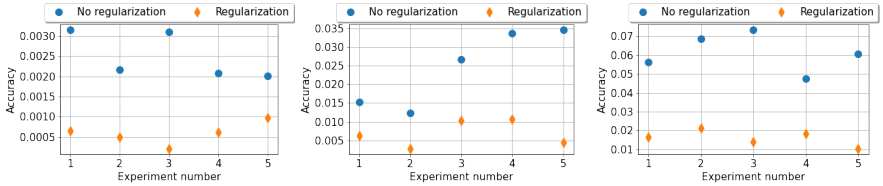
Consider a Hamiltonian system with Hamiltonian function $H : \mathbb{R}^{2n} \to \mathbb{R}$, and a functionally independent first integral G, i.e. $\nabla H(x)$ and $\nabla G(x)$ are never parallel. Consider the numerical integration $\hat{y}_k^j,\ j = 1, ..., M$ of the approximated Hamiltonian vector field X_{H_Θ}, starting at $\hat{y}_k^1 = x_k$. In the ideal case in which the learned Hamiltonian H_Θ coincides with H and the numerical flow is replaced with the exact one, both H and G should be conserved. For this reason, we suggest adding to the loss function in equation (4.3.2) the following "regularization" term:

$$\mu \sum_{j \in \mathcal{I}} \left(G(\hat{y}_k^j) - G(x_k) \right)^2$$

for all the training points x_k. Here $\mathcal{I}$ is a subset of indices contained in $\{1, ..., M\}$, and μ is a regularization parameter that balances the importance of the preservation of the additional first integral against the perfect fitting of the training trajectories. We test this regularization procedure with the Hamiltonian system $X_H \in \mathfrak{X}(\mathbb{R}^4)$ defined by

$$H(q_1, q_2, p_1, p_2) = \frac{q_1^2 + p_1^2}{2} + \frac{p_2^2}{2} + \frac{1}{2} q_2^2 + \frac{1}{4} q_2^4 = h_1(q_1, p_1) + h_2(q_2, p_2).$$

This system has $G(q, p) := h_1(q_1, p_1)$ as an additional independent first integral other than H. We report in Figure 4.3 some plots of the obtained $\mathcal{E}_1$ values as defined in (4.1.2). In these experiments we add some random noise of the form $\varepsilon\delta$ to the points y_i^j of the numerical trajectories, where $\delta \sim \mathcal{N}(0, 1)$ follows a standard normal distribution. The same experiment is run 5 times, and for each of these we plot the obtained $\mathcal{E}_1$ value. For each experiment we generate new training and test trajectories, and these are used for both the regularized training and the non regularized one. Furthermore, each experiment has a different random initialization of the weights, which is however shared between the regularized and non regularized networks. We notice that with regularization we can consistently get a better error in terms of the $\mathcal{E}_1$ measure. There is not a huge difference between the results, however. This suggests that when prior information is known, it might be important to experiment with this kind of regularizing terms. To conclude the Section, we highlight how remarkable it is that even without the regularization term, the trajectories are qualitatively well captured by the network and hence the test error is quite low. This is mostly due to the prior physical knowledge we impose on the learning procedure, i.e. that the vector field should be Hamiltonian. Indeed, since in the worst case the network approximates the wrong Hamiltonian, we always expect that it does not overfit the noisy trajectories, since they can not be learned exactly. On



(a) Case $\varepsilon = 0.1$ **(b)** Case $\varepsilon = 0.3$ **(c)** Case $\varepsilon = 0.5$

Figure 4.3: 5 repeated experiments for each perturbation regime. We plot on the y axis the average accuracy, in terms of the $\mathcal{E}_1$ measure, obtained with the trained network, when compared with the real (non-noisy) trajectories.

the other hand, without the prior knowledge of the Hamiltonian nature of the system, all the overfitting problems of standard neural networks reoccur and the risk of being closer to an interpolant of the noisy trajectories is higher.

4.4 Learning constrained Hamiltonian systems

The approximation of the Hamiltonians of constrained mechanical systems with neural networks has already been studied in the literature. Two main approaches can be identified. One of them is based on local coordinates on the constrained manifold (see, e.g., [9], [14]) and the other uses ambient space coordinates and Lagrange multipliers (see [13]). In principle, both the formulations apply to any constrained Hamiltonian system. However, as remarked in [13], the choice of a redundant system of coordinates usually gives a simpler expression for the Hamiltonian. This results in a more data efficient training procedure. In the second approach an embedded Runge–Kutta pair of order (5,4) is used to train the network. This choice inevitably leads to a drift from the constrained manifold during the training, even if it can be reduced by setting the tolerances of the integrator. However, in this way the cost of the integrator increases, hence this is not the most efficient way to preserve the constraints.

In this work, we use an alternative global formulation of the dynamics, as introduced in Section 4.2. In principle this formulation adapts to any constrained Hamiltonian system whose configuration manifold is a submanifold of $\mathbb{R}^n$. Coupling this description of the dynamics with the learning framework introduced in Section 4.3, their Hamiltonian functions can be approximated. To be more precise, one can use any numerical integrator to discretize the constrained trajectories and compare them with the training data. For example, Runge–Kutta 4 method can be used and this experimentally gives fast training procedures and accurate approximations of the Hamiltonian, as shown in the experiments of Subsection 4.4.3.

We remark that in general numerical integrators do not preserve the geometry of the system and there might be a drift from the constrained manifold (see, e.g., [15, Chapter 7]). Experimentally this does not seem to have a great impact on the quality of the predicted Hamiltonian in most of the cases. However, as we present in the numerical experiments with Lie group integrators, there might be situations in which one benefits from training the Hamiltonian with an integrator preserving the phase space. Notice that the Hamiltonian that defines the dynamics has non-unique extension outside the phase space $\mathcal{M} = T^*\mathcal{Q}$. This is due to the projection matrix $P(q) = I_n - G(q)\left(G(q)^T G(q))\right)^{-1} G(q)^T$ appearing Equation (4.2.1), where $G(q)$ is the Jacobian matrix of the constraint function $g(q)$ defining $\mathcal{Q}$. This justifies investigating the importance of the preservation of the manifold T^*Q in the training procedure.

As introduced in Section 4.2, in this work we assume that the constrained configuration manifold $\mathcal{Q}$ is known. Referring to equation (4.2.1), we notice that once the geometry is known, it is enough to specify the Hamiltonian function $H : T^*\mathcal{Q} \subseteq \mathbb{R}^{2n} \to \mathbb{R}$ in order to characterize the dynamics of a system. We show a setting in which the geometry can be preserved by Lie group integrators (see, [18], [6], [8]) focusing on the case $T^*\mathcal{Q}$ is homogeneous[2]. We see this even as an opportunity to study the behaviour of this class of methods in an applied framework and combined with neural networks. This geometric setup applies, for example, when $\mathcal{Q}$ is a homogeneous manifold and the transitive action $\psi : G \times \mathcal{Q} \to \mathcal{Q}$ defines, for any $q \in \mathcal{Q}$, a submersion $\psi_q : G \to \mathcal{Q}$ at the identity element $e \in G$ (see, e.g., [2], [7]). Cartesian products of homogeneous manifolds are homogeneous too. Usually, multibody systems have constrained configuration manifolds given by cartesian products of S^2, $\mathbb{R}^k$, $SO(3)$ and $SE(3)$, which are respectively the special orthogonal and Euclidean groups. These are all homogeneous manifolds and so are their tangent and cotangent bundles.

4.4.1 Lie group methods and neural networks

Among the various classes of Lie group methods, we consider the Runge–Kutta–Munthe–Kaas (RKMK) methods and the commutator free ones (see, e.g., [22], [5]). The underlying idea of RKMK methods, applied to $F \in \mathfrak{X}(\mathcal{M})$, with $\mathcal{M}$ an arbitrary homogeneous manifold, is to express F as $F|_m = \psi_*(f(m))|_m$. Here ψ_* is the infinitesimal generator of ψ, a transitive Lie group action of G on $\mathcal{M}$, and $f : \mathcal{M} \to \mathfrak{g}$ is a function that locally lifts the dynamics to the Lie algebra $\mathfrak{g}$ of G. On this linear space, we can perform a time step integration. We then map the result back to $\mathcal{M}$, and repeat this up to the final integration time. More

[2] A smooth manifold $\mathcal{M}$ is homogeneous if for any pair of points $m_1, m_2 \in \mathcal{M}$ there is $g \in G$ such that $\psi(g, m_1) = m_2$, where $\psi : G \times \mathcal{M} \to \mathcal{M}$ is a Lie group action. In other words, ψ is a transitive action.

explicitly, let Δt be the size of the uniform time step of the discretization, we then update $y_n \in \mathcal{M}$ to y_{n+1} by

$$\begin{cases} \gamma(0) = 0 \in \mathfrak{g}, \\ \dot{\gamma}(t) = \operatorname{dexp}^{-1}_{\gamma(t)} \circ f \circ \psi(\exp(\gamma(t)), y_n) \in T_{\gamma(t)}\mathfrak{g}, \\ y_{n+1} = \psi(\exp(\gamma_1), y_n) \in \mathcal{M}, \end{cases} \tag{4.4.1}$$

where $\gamma_1 \approx \gamma(\Delta t) \in \mathfrak{g}$ is computed with a Runge–Kutta method, and dexp^{-1} is the inverse of the differential of the exponential map $\exp : \mathfrak{g} \to G$ as defined, for example, in [18, Section 2.6]. We do not go into the details of commutator free methods, but the following development applies to them as well. In particular the function f still plays a fundamental role.

We now present a natural way to combine the learning framework typical of unconstrained systems with Lie group integrators. This is done introducing a Lie group method during the learning procedure. Indeed, since we want to apply a Lie group integrator to deal with nonlinear geometries, we set $\Psi^{\Delta t}$, defined in equation (4.3.1), to be the Δt update given by some RKMK method. In other words, using the notation of equation (4.4.1), we get $\Psi^{\Delta t}(z) = \psi(\exp(\gamma_1), z)$ with $\gamma_1 \in \mathfrak{g}$.

The setting presented above for generic vector fields on homogeneous manifolds simplifies considerably in the presence of Hamiltonian systems. Indeed, for this type of systems, what is needed to fully determine the dynamics is the geometry given by $\mathcal{M} = T^*\mathcal{Q}$ and the scalar Hamiltonian function $H : \mathcal{M} \to \mathbb{R}$. In other words, we can think of the function $f : \mathcal{M} \to \mathfrak{g}$, that allows to express the vector field in terms of the infinitesimal generator of the action, as the result of an operator $F : C^1(\mathcal{M}, \mathbb{R}) \to \{T^*\mathcal{M} \to \mathfrak{g}\}$ acting on a scalar function H. More explicitly, we can write $f = F[H]$ where F and H encode respectively the geometry and the dynamics of the system. This operator is not really necessary, but it clarifies considerably how the neural network comes into play in the learning framework. Indeed, because of this construction, we can write the numerical flow $\Psi^{\Delta t}$ as the map sending y_n into $y_{n+1} = \psi(\exp(\gamma_{\Delta t, y_n}), y_n)$ with $\gamma_{\Delta t, y_n}$ being an approximation of the solution $\gamma(\Delta t)$ of the following initial value problem

$$\begin{cases} \dot{\gamma}(t) = \operatorname{dexp}^{-1}_{\gamma(t)} \circ F[H_\Theta] \circ \psi(\exp(\gamma(t)), y_n) \in T_{\gamma(t)}\mathfrak{g}, \\ \gamma(0) = 0 \in \mathfrak{g}. \end{cases}$$

Here H_Θ is the approximation of the Hamiltonian given by the current weights Θ of the neural network. Thus, applying a particular family of geometric numerical integrators, we can directly study some constrained systems with the same ideas coming from learning unconstrained ones. Since following this procedure

the geometry is preserved, one can consider replacing the Euclidean distance in the loss function defined in equation (4.3.2) with a Riemannian metric of the constrained manifold. This would bring to distances between points that correspond to the length of the minimal geodesic connecting them, which is in general different from the length of the segment in the ambient space having them as extrema. In the remaining part of the Section, we specialize this reasoning to mechanical systems defined on copies of T^*S^2. We focus on a chain of spherical pendula, but the geometric setting applies also to other systems (see, e.g., [20, Section 10.5]).

4.4.2 Mechanical systems on $(T^*S^2)^k$

As anticipated in the introductory Section, in this geometric setting we are not involving symplectic integrators and we do not assume to have a separable Hamiltonian anymore. Thus, we now model a more general family of Hamiltonians as

$$H(q,p) = \frac{1}{2} p^T M^{-1}(q)\, p + V(q). \tag{4.4.2}$$

We model the potential energy as before, however we need an alternative strategy for the inverse of the mass matrix, which is no longer assumed to be constant. Based on the problem, one can choose various parametrizations of the mass matrix or its inverse. We decide to specialize the architecture based on the fact that the geometry of the system is known to be $\mathcal{M} = (T^*S^2)^k$, where $S^2 \subset \mathbb{R}^3$. We coordinatize $\mathcal{M}$ with $(q,p) = (q_1,\dots,q_k,p_1,\dots,p_k) \in \mathbb{R}^{6k}$. In this case, when $p \in \mathbb{R}^{3k}$ is intended as the vector of linear momenta, the matrix $M(q)$ in equation (4.4.2) is a block matrix, with

$$i,j = 1,...,k, \quad \mathbb{R}^{3\times 3} \ni M(q)_{ij} = \begin{cases} m_{ii} I_3, & i = j \\ m_{ij}(I_3 - q_i q_i^T), & \text{otherwise,} \end{cases}$$

see [20, Section 8.3.3] for further details. Here, the matrix having constant entries m_{ij} is symmetric and positive definite. For this reason, we leverage this form of the kinetic energy and learn a constant matrix $A \in \mathbb{R}^{k\times k}$ and a vector $b \in \mathbb{R}^k$ so that

$$\begin{bmatrix} m_{11} & ... & m_{1k} \\ m_{21} & ... & m_{2k} \\ \vdots & \vdots & \vdots \\ m_{k1} & ... & m_{kk} \end{bmatrix} \approx A^T A + \begin{bmatrix} \tilde{b}_1 & 0 & ... & 0 \\ 0 & \tilde{b}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & ... & 0 & \tilde{b}_k \end{bmatrix} \tag{4.4.3}$$

where $\tilde{b}_i := \max(0, b_i)$ are terms added to promote the positive definiteness of the right-hand side. We tested also elevating to the second power the b_i

instead of taking the maximum with 0, but we got better results with the choice presented in equation (4.4.3). The matrix on the left-hand side of equation (4.4.3) is exactly the one appearing in Hamiltonian formulations with Cartesian coordinates, as the one used in [13].

For the spherical pendulum we have $k = 1$ and hence the Hamiltonian dynamics is defined on its cotangent bundle T^*S^2, which is a homogeneous manifold. This can be obtained thanks to the transitivity of the group action

$$\Psi : SE(3) \times T^*S^2 \to T^*S^2,\ ((R, r), (q, p^T)) \mapsto (Rq, (Rp + r \times Rq)^T),$$

where the transpose comes from the usual interpretation of covectors as row vectors. As in [16, Chapter 6], we represent a generic element of the special Euclidean group $G = SE(3)$ as an ordered pair (R, r), where $R \in SO(3)$ is a rotation matrix and $r \in \mathbb{R}^3$ is a vector. With this specific choice of the geometry, the formulation presented in equation (4.2.1) simplifies considerably. Indeed $P(q) = I_3 - qq^T$ which implies $W(q, p) = pq^T - qp^T$. Replacing these expressions in (4.2.1) and using the triple product rule we end up with the following set of ODEs

$$\begin{cases} \dot{q} &= (I - qq^T)\partial_p H(q, p) \\ \dot{p} &= -(I - qq^T)\partial_q H(q, p) + \partial_p H(q, p) \times (p \times q). \end{cases} \tag{4.4.4}$$

This vector field $X(q, p)$ can be expressed as $\psi_*(F[H](q, p))(q, p)$ with

$$\psi_*((\xi, \eta))(q, p) = (\xi \times q, \xi \times p + \eta \times q), \quad (\xi, \eta) \in \mathfrak{g} = \mathfrak{se}(3)$$

and

$$F[H](q, p) = (\xi, \eta) = \left(q \times \frac{\partial H(q, p)}{\partial p}, \frac{\partial H(q, p)}{\partial q} \times q + \frac{\partial H(q, p)}{\partial p} \times p \right).$$

A similar reasoning can be extended to a chain of k connected pendula, and hence to a system on $(T^*S^2)^k$. The main idea is to replicate both the equations (4.4.4) and the expression $F[H]$ for all the k copies of T^*S^2. A more detailed explanation can be found in [8].

We present in Figure 4.4 the results obtained for the training of a double pendulum, i.e. $k = 2$. To train the network, we generate a set of $N = 500$ training trajectories with the embedded Runge–Kutta pair of order (5,4) of `SciPy`. The final integration time is $T = 0.1$ and $M = 5$. To model the potential energy, we use a feedforward network with 3 hidden layers of 100 neurons each. In the plots we show the configuration variables, $q_1, q_2 \in S^2$, obtained for 100 test trajectories in the time interval $[0, 1]$, where the network H_Θ has been trained with a commutator free method of order 4.

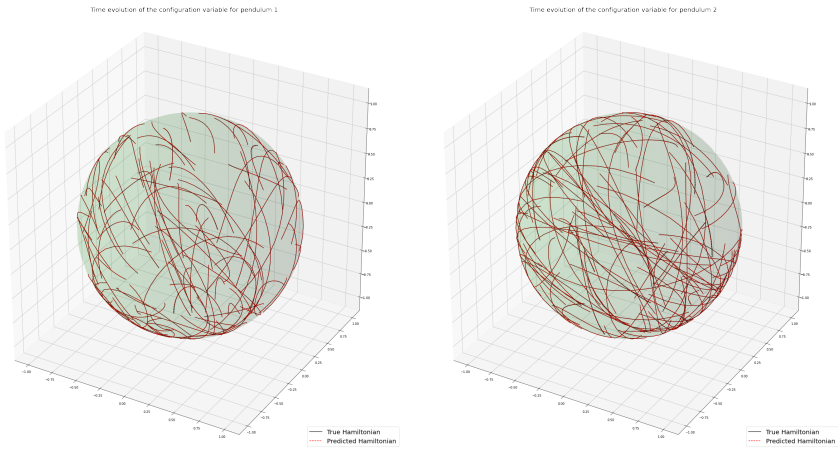


Figure 4.4: Comparison between 100 test trajectories obtained with the true Hamiltonian H and the predicted one H_Θ. To train H_Θ, a Lie group method is used. This gives $\mathcal{E}_1 = 2.65 \cdot 10^{-6}$ and a final training loss of $1.6 \cdot 10^{-9}$.

4.4.3 Experimental study of the learning procedure

We investigate the influence of the training setup on the error measures $\mathcal{E}_1$, $\mathcal{E}_2$, defined in (4.1.3), and on the training loss. More precisely, we test how the parameters M, N, the noise magnitude and the training integrator affect the performance of the network. We quantify the magnitude of noise in the training trajectories with a parameter $\varepsilon > 0$, as in Subsection 4.3.2. The integrators that we study are Lie Euler, explicit Euler (both of order 1), commutator free and Runge–Kutta (both of order 4). In particular, Lie Euler and commutator free methods preserve the phase space $\mathcal{M}$ up to machine accuracy. To get a sufficient sample of experiments, we repeat all the tests 5 times, and look at the medians and geometric means[3] of the obtained results. To be precise, we test $N \in \{50, 500, 1000, 1500\}$, $M \in \{2, 3, 5\}$, and $\varepsilon \in \{0, 0.001, 0.01, 0.1\}$. Therefore, we perform a total of 960 experiments, and also here the potential energy is modelled with a feedforward network of 3 hidden layers having 100 neurons each. Furthermore, for the four experiments performed varying just the integrator, and with the other parameters fixed, the network's weights are initialized to be the same, and also the training and test initial conditions are the same. For all these experiments, we focus on the single spherical pendulum, we keep the final training time to $T = 0.1$, and we don't use regularization terms. The training trajectories have been generated with the `SciPy` imple-

[3]The choice of geometric means is because of the exponential nature of the error measures and the training loss.

mentation of the Dormand-Prince pair of order (5,4) with strict tolerance.

Order	Integrator	$\mathcal{E}_1$	$\mathcal{E}_2$	Training Loss
1	EE	5.7e-5	1.13e-2	2.12e-6
1	LE	4.9e-5	1.07e-2	1.17e-6
4	RK4	1.12e-5	3.83e-3	2.63e-7
4	CF4	1.12e-5	3.85e-3	2.64e-7

Table 4.1: In this table we report the geometric means of the quantities $\mathcal{E}_1$, $\mathcal{E}_2$ and the training loss. Here we average over all the 240 experiments that have the same integrator. We denote the four integrators with EE (explicit Euler), LE (Lie Euler), RK4 (Runge–Kutta 4), and CF4 (commutator free 4).

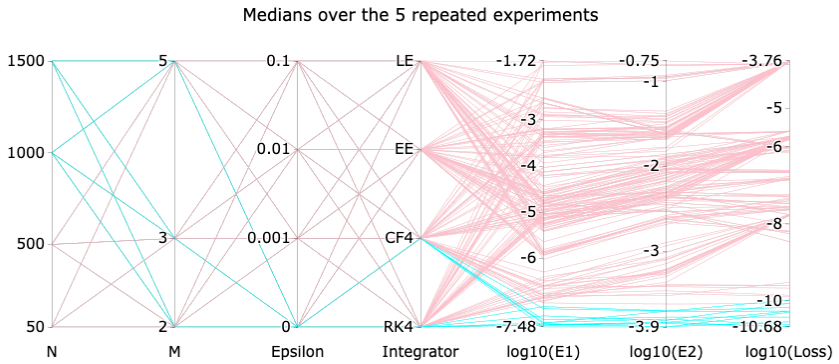


Figure 4.5: This is a parallel coordinate plot reporting the dependencies of $\mathcal{E}_1$, $\mathcal{E}_2$ and the training loss on the parameters N, M, ε and on the integrator. Each coloured polyline corresponds to the median over 5 experiments, i.e. same N, M, ε and same integrator. The lines in cyan color represent the combinations giving $\mathcal{E}_1 < 10^{-7}$.

As shown in Table 4.1, the order of the numerical integrator used to train the network plays an important role. Indeed, we get results that are similar for methods of the same order, but there is a noticeable decay in the errors and in the loss when we increase the order from one to four. As highlighted in [27], this effect can be explained with a standard argument of backward error analysis, see e.g. [15, Chapter 9]. From the results reported in Table 4.1 we see that the local error of the integrator is more important than the preservation of the geometry. Therefore, even if from a theoretical point of view it seems relevant to remain on the manifold during the training, in practice this does not seem to be very important in the particular experiment considered here. In Figure 4.5, we plot the dependencies of $\mathcal{E}_1$, $\mathcal{E}_2$ and the training loss, on N, M, ε and the integrator. We notice that values of $\mathcal{E}_1$ below a threshold of 10^{-7}

Without noise									
Integrator of order 1					Integrator of order 4				
N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$	N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$
1500	5	LE	1e-6	2.4e-3	1500	2	CF4	3.2e-8	1.3e-4
1000	5	LE	1e-6	2.3e-3	1500	5	RK4	3.3e-8	1.4e-4
1000	5	EE	1e-6	2.4e-3	1500	3	RK4	3.4e-8	1.4e-4
1500	5	EE	1e-6	2.5e-3	1500	2	RK4	3.6e-8	1.5e-4
500	5	LE	2e-6	2.6e-3	1500	3	CF4	3.7e-8	1.5e-4
With noise									
Integrator of order 1					Integrator of order 4				
N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$	N	M	Int.	$\mathcal{E}_1$	$\mathcal{E}_2$
1500	5	LE	1.2e-5	6.6e-3	1500	5	RK4	5e-6	3.4e-3
1500	5	EE	1.2e-5	6.3e-3	1500	5	CF4	6e-6	4.1e-3
1000	5	LE	1.5e-5	6.6e-3	1000	5	CF4	7e-6	3.8e-3
1000	5	EE	1.6e-5	6.8e-3	1000	5	RK4	8e-6	4.3e-3
500	5	LE	1.8e-5	6.7e-3	1000	3	RK4	8e-6	4.2e-3

Table 4.2: In this Table we report the combinations that give the 5 best values of $\mathcal{E}_1$, together with the corresponding value $\mathcal{E}_2$. These are the geometric means among all the experiments. The two tables compare the performance on data with and without noise.

can be reached only with integrators of order four and with the smallest value of ε. The interplay of N, M and ε is further investigated in Table 4.2. An interactive version of Figure 4.5, together with other parallel coordinate plots, can be found at the GitHub Page `https://davidemurari.github.io/learningConstrainedHamiltonians/`, while the dataset is available in the GitHub repository associated to the paper.

We conclude this parameter study considering separately the case with and without noise, $\varepsilon > 0$ and $\varepsilon = 0$ respectively. The results are reported in Table 4.2. In general the lowest values of $\mathcal{E}_1$ are obtained with high N. For the model under consideration, $N = 1000$ seems already high enough to achieve good results. Regarding M, Table 4.2 shows that to achieve lower values of $\mathcal{E}_1$ in the presence of noise, one needs to adopt a higher M. On the other hand, in the absence of noise it seems important to have a high M only for low order integrators. Finally, as may be expected, even if this Table does not distinguish among the different magnitudes of the noise, we see that with $\varepsilon = 0$ better results can be achieved.

We also point out that the experiments were performed for short integration times, where not only symplectic integrators can generate physically meaningful trajectories. It would be interesting to explore the performance of sympelctic

and constraint preserving integrators in this setting (see, e.g., [1]) and we defer this to further work.

Besides the theoretical aspect of the non-uniqueness of the extension of the dynamics outisde of $\mathcal{M} \subset \mathbb{R}^{2n}$, we now report a numerical experiment where the preservation of the geometry during the training is beneficial. We consider again a simple spherical pendulum and we assume to know that the potential energy is linear. We hence impose this prior information on the architecture of the network. Due to the problem's simplicity, we aim to reach very low $\mathcal{E}_1$ and $\mathcal{E}_2$ values. Training the same architecture for 200 epochs, both with Runge–Kutta and commutator free methods of order 4, we get the results in Table 4.3. Indeed the geometric integrator outperforms the classical Runge-Kutta method in this experiment.

Numerical method in the training	$\mathcal{E}_1$	$\mathcal{E}_2$
Runge-Kutta of order 4	4.2e-12	1.5e-6
Commutator free of order 4	1.1e-14	2.5e-7

Table 4.3: Comparison of the accuracy measures $\mathcal{E}_1$ and $\mathcal{E}_2$ obtained with the two integrators. These results are obtained imposing the linear structure of the potential energy on the network modelling the Hamiltonian of the spherical pendulum. The kinetic energy has been modelled as in previous experiments.

This experiment suggests that the choice of an integrator that does not fully exploit the available information, like the geometry, might limit the quality of the obtained approximations. For those cases in which one is interested in as accurate as possible predictions, this might be a relevant issue.

The experiments performed lead to the conclusion that modelling multi-body systems with neural networks can be a valuable approach. However, to better leverage the approximation capabilities of machine learning techniques (see, e.g., [17], [10]) we believe that a deeper investigation and understanding of how they interface with physical models is necessary.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124.

Bibliography

[1] Hans C Andersen. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *Journal of computational Physics*, 52(1):24–34, 1983. 104

[2] RW Brockett and HJ Sussmann. Tangent bundles of homogeneous spaces are homogeneous spaces. In *Proc. Amer. Math. Soc*, volume 35, pages 550–551, 1972. 76, 97

[3] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016. 17, 92

[4] Elena Celledoni, Matthias Joachim Ehrhardt, Christian Etmann, Brynjulf Owren, Carola-Bibiane Schonlieb, and Ferdia Sherry. Equivariant neural networks for inverse problems. *Inverse Problems*, 37, 2021. 92

[5] Elena Celledoni, Arne Marthinsen, and Brynjulf Owren. Commutator-free Lie group methods. *Future Generation Computer Systems*, 19(3):341–352, 2003. 97

[6] Elena Celledoni, Håkon Marthinsen, and Brynjulf Owren. An introduction to Lie group integrators–basics, new developments and applications. *Journal of Computational Physics*, 257:1040–1061, 2014. 2, 97

[7] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Dynamics of the N-fold Pendulum in the framework of Lie Group Integrators. *arXiv preprint arXiv:2109.12325*, 2021. 97

[8] Elena Celledoni, Ergys Çokaj, Andrea Leone, Davide Murari, and Brynjulf Owren. Lie group integrators for mechanical systems. *International Journal of Computer Mathematics*, 0(0):1–31, 2021. 75, 79, 85, 97, 100

[9] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *International Conference on Learning Representations*, 2020. 85, 86, 90, 94, 96

[10] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 104

[11] Marco David and Florian Méhats. Symplectic Learning for Hamiltonian Neural Networks. *arXiv preprint arXiv:2106.11753*, 2021. 87

[12] Daniel M. DiPietro, Shiying Xiong, and Bo Zhu. Sparse Symplectically Integrated Neural Networks. In *Advances in Neural Information Processing Systems 34*. 2020. 92

[13] Marc Finzi, Alex Wang, and Andrew Gordon Wilson. Simplifying Hamiltonian and Lagrangian Neural Networks via Explicit Constraints. *NeurIPS*, 2020. 14, 85, 89, 96, 100

[14] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. *Advances in Neural Information Processing Systems*, 32:15379–15389, 2019. 2, 85, 86, 96

[15] Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006. 86, 97, 102

[16] Darryl D Holm. *Geometric Mechanics-Part II: Rotating, Translating and Rolling*. World Scientific, 2011. 100

[17] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991. 104

[18] Arieh Iserles, Hans Z Munthe-Kaas, Syvert P Nørsett, and Antonella Zanna. Lie-group methods. *Acta numerica*, 9:215–365, 2000. 5, 14, 16, 97, 98

[19] John M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer New York, NY, 2012. 3, 86, 89

[20] T. Lee, M. Leok, and N. H. McClamroch. *Global formulations of Lagrangian and Hamiltonian dynamics on manifolds*. Interaction of Mechanics and Mathematics. Springer, Cham, 2018. 1, 3, 4, 5, 6, 85, 88, 99

[21] Jerrold E Marsden and Tudor S Ratiu. Introduction to mechanics and symmetry. *Physics Today*, 48(12):65, 1995. 86

[22] H. Munthe-Kaas. High order Runge–Kutta methods on manifolds. *Appl. Num. Math.*, 29:115–127, 1999. 32, 33, 34, 37, 65, 97

[23] Christian Offen and Sina Ober-Blöbaum. Symplectic integration of learned Hamiltonian systems. *arXiv preprint arXiv:2108.02492*, 2021. 85

[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985. 90

[25] E. T. Whittaker. *A treatise on the analytical dynamics of particles and rigid bodies*. Cambridge University Press, 1993. Fourth edition. 86

[26] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019. 85, 86, 93

[27] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep Hamiltonian networks based on symplectic integrators. *arXiv preprint arXiv:2004.13830*, 2020. 90, 102

Neural networks for the approximation of Euler's elastica

Elena Celledoni, Ergys Çokaj, Andrea Leone,
Sigrid Leyendecker, Davide Murari, Brynjulf Owren,
Rodrigo T. Sato Martín de Almagro and Martina Stavole

Submitted

Neural networks for the approximation of Euler's elastica

Abstract. Euler's elastica is a classical model of flexible slender structures, relevant in many industrial applications. Static equilibrium equations can be derived via a variational principle. The accurate approximation of solutions of this problem can be challenging due to nonlinearity and constraints. We here present two neural network based approaches for the simulation of this Euler's elastica. Starting from a data set of solutions of the discretised static equilibria, we train the neural networks to produce solutions for unseen boundary conditions. We present a *discrete* approach learning discrete solutions from the discrete data. We then consider a *continuous* approach using the same training data set, but learning continuous solutions to the problem. We present numerical evidence that the proposed neural networks can effectively approximate configurations of the planar Euler's elastica for a range of different boundary conditions.

5.1 Introduction

Modelling of mechanical systems is relevant in various branches of engineering. Typically, it leads to the formulation of variational problems and differential equations, whose solutions are approximated with numerical techniques. The efficient solution of linear and non-linear systems resulting from the discretisation of mechanical problems has been a persistent challenge of applied mathematics. While classical solvers are characterised by a well-established and mature body of literature [3, 13, 14, 26, 30, 33, 36], the past decade has witnessed a surge in the use of novel machine learning-assisted techniques [4, 5, 7, 8, 10, 12, 16, 20, 21, 23, 24, 28, 29, 34, 38, 39, 46]. These approaches aim at enhancing solution methods by leveraging the wealth of available data and known physical principles. The use of deep learning techniques to improve the performance of traditional numerical algorithms in terms of efficiency, accuracy, and computational scalability, is becoming increasingly popular also in computational mechanics. Examples comprise virtually any problem where approximation of functions is required, but also efficient reduced order modelling e.g. in fluid mechanics, the deep Ritz method, or more specific numerical tasks such as optimisation of the quadrature rule for the computation of the finite element stiffness matrix, acceleration of simulations on coarser meshes by learning appropriate collocation points, and replacing expensive numerical computations with data-driven predictions [4, 18, 44, 46, 47]. This recent literature is evidence that neural networks can be used successfully as surrogate models for the solution operators of various differential equations.

In the context of ordinary and partial differential equations, two main trends can be identified. The first one aims at providing a machine learning based

approximation to the discrete solutions of differential problems on a certain space-time grid, for example by solving linear or nonlinear systems efficiently and accelerating convergence of iterative schemes [5,12,16,20,21]. The second one provides instead solutions to the differential problem as continuous (and differentiable) functions of the temporal and spatial variables. Depending on the context, conditions on such approximate solutions are then provided by the differential problem itself, by the initial values and the boundary conditions, and by the available data. The idea of providing approximate solutions as functions defined on the space-time domain and parametrised as neural networks was proposed in the nineties [19] and was recently revived in the framework of Physics-Informed Neural Networks in [34]. Since then, such an approach has attracted a lot of interest and has developed in many directions [7, 18, 39].

In this work, we use neural networks to approximate the configurations of highly flexible slender structures modelled as beams. Such models are of great interest in industrial applications like cable car ropes, diverse types of wires or endoscopes [25, 31, 37, 41]. Notwithstanding their ingenious and simple mathematical formulation, slender structure models can accurately reproduce complex mechanical behaviour and for this reason their numerical discretisation is often challenging. Furthermore, the use of 3-dimensional models requires high computational time. Due to the fact that slender deformable structures have one dimension (length) being orders of magnitude larger than their other dimensions (cross-section), it is possible to reduce the complexity of the problem from a 3-dimensional elastic continuum to a 1-dimensional beam. A beam is modelled as a centerline curve, $\mathbf{q} : [0, L] \to \mathbb{R}^n, s \mapsto \mathbf{q}(s)$, with $n = 2$ or $n = 3$, along which a rigid cross-section $\Sigma(s)$ is attached. The main model assumption is that the diameter of $\Sigma(s)$ is small compared with the undeformed length L. We here consider a special case of a beam where the cross-section $\Sigma(s)$ is constant and orthogonal to the centerline, the 2-dimensional *Euler's elastica* [9]. In this case, $\mathbf{q}(s)$ is inextensible with fixed boundary conditions and is the solution of a bending energy minimisation problem [22, 27, 40].

When approximating static equilibria of the Euler's elastica via neural networks, a key issue is to ensure the inextensibility of the curve (having unit norm tangents) as well as the boundary conditions. Two main approaches can be found in the literature [18, 35, 39]. One is the weak imposition of constraints and boundary conditions adding appropriate, extra terms to the loss function. The other is a strong imposition strategy consisting in shaping the network architectures so that they satisfy the constraints by construction. We show examples of both the approaches in Sections 5.4 and 5.5.

The paper is organised as follows. In Section 5.2, we present the mathematical model of the planar Euler's elastica, including its continuous and discrete equilibrium equations. We describe the approach used to generate the data sets

for the numerical experiments. In Section 5.3, we introduce some basic theory and notation for neural networks that we shall use in the succeeding sections. Starting from general theory, we specialise in the task of approximating configurations of the Euler's elastica. In Section 5.4, we introduce the *discrete* approach, which aims to approximate precomputed numerical discretisations of the Euler's elastica. We discuss some drawbacks associated with this approach and then propose an alternative approximation strategy in Section 5.5. The *continuous* approach consists in computing an arc length parametrisation of the beam configuration. We provide insights into two additional networks and analyse how the test accuracy changes with varying constraints, such as boundary conditions or tangent vector norms.

Main contributions: This paper presents advancements in the approximation of beam configurations using neural networks. These advancements include: (i) An extensive experimental analysis of approximating numerical discretisations of Euler's elastica configurations through what we call *discrete networks*, (ii) Identification and discussion of the limitations associated with this discrete approach, and (iii) Introduction of a new parametrisation strategy called *continuous network* to address some of these drawbacks.

5.2 Euler's elastica model

We consider an inextensible beam model in which the cross-section $\Sigma(s)$ is assumed to be constant along the arc length s and perpendicular to the centerline $\mathbf{q}(s)$, which means that no shear deformation can occur. Thus, the deformation of the centerline is a pure bending problem, precisely the Euler's elastica curve. In the following, we assume $\mathbf{q} \in C^2\left([0, L], \mathbb{R}^2\right)$, i.e., the curve is planar and twice continuously differentiable with length L. If s denotes the arc length parameter, then $\left\|\mathbf{q}'(s)\right\| = 1$, where $' = \frac{d}{ds}$, for all $s \in [0, L]$. The elastica problem consists in minimising the following Euler-Bernoulli energy functional

$$\int_0^L \kappa(s)^2 \, ds,$$

where $\kappa(s)$ denotes the curvature of $\mathbf{q}(s)$, [27]. Given the arc length parametrisation, then $\kappa(s) = \left\|\mathbf{q}''(s)\right\|$.

We can reformulate this problem as a constrained Lagrangian problem as follows. Consider the second-order Lagrangian $\mathcal{L} : T^{(2)}Q \to \mathbb{R}$, where $T^{(2)}Q$ denotes the second-order tangent bundle [6] of the configuration manifold Q, which in this case is $\mathbb{R}^2$:

$$\mathcal{L}\left(\mathbf{q}, \mathbf{q}', \mathbf{q}''\right) = \frac{1}{2} EI \left\|\mathbf{q}''\right\|^2 . \tag{5.2.1}$$

Nomenclature	
$\mathcal{L}$	continuous Lagrangian function
$\mathcal{S}$	continuous action functional
$\mathcal{L}_d$	discrete Lagrangian function
$\mathcal{S}_d$	discrete action functional
$\mathbf{q}$	configuration of the beam
$\mathbf{q}'$	first spatial derivative of $\mathbf{q}$
θ	tangential angle
s	arc length parameter
κ	curvature
L	length of the undeformed beam
EI	bending stiffness, with E the elastic modulus and I the second moment of area
$\hat{\mathbf{q}}$	numerical approximation of $\mathbf{q}$
$N+1$	number of discretisation nodes, with N the number of intervals
h	space step (length of each interval)
$q^{\mathrm{d}}_{\boldsymbol{\rho}}$	discrete neural network
$q^{\mathrm{c}}_{\boldsymbol{\rho}}$	continuous neural network approximating the solution curve $\boldsymbol{q}(s)$
$\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$	continuous neural network approximating the angular function $\theta(s)$
$\boldsymbol{\rho}$	parameters of the neural network
ℓ	number of layers in the neural network
σ	activation function
M	number of training data
B	size of one training batch
MSE	mean squared error
MLP	multi layer perceptron
ResNet	residual neural network
MULT	multiplicative neural network
$\mathcal{D}$	differential operator
$\mathcal{I}$	quadrature operator

Table 5.1: List of abbreviations and notations.

Here, abusing the notation, $'$ denotes a spatial derivative, but we do not initially assume arc length parametrisation. The parameter EI is the bending stiffness, which governs the response of the elastica under bending. This mechanical parameter consists of a material and a geometric properties, where E is the Young's modulus and I is the second moment of area of the cross-section Σ. For simplicity, these parameters are assumed to be constant along the length of the beam.

In order to recover the solutions of the elastica, the Lagrangian in Equation (5.2.1) must be supplemented with the constraint equation

$$\Phi\left(\mathbf{q}, \mathbf{q}'\right) = \left\|\mathbf{q}'\right\|^2 - 1 = 0. \tag{5.2.2}$$

This imposes arc length parametrisation of the curve $\mathbf{q}(s)$ and leads to the augmented Lagrangian $\widetilde{\mathcal{L}}\colon T^{(2)}Q\times\mathbb{R}\to\mathbb{R}$

$$\widetilde{\mathcal{L}}\left(\mathbf{q},\mathbf{q}',\mathbf{q}'',\Lambda\right)=\mathcal{L}\left(\mathbf{q},\mathbf{q}',\mathbf{q}''\right)+\Lambda\Phi\left(\mathbf{q},\mathbf{q}'\right), \tag{5.2.3}$$

where $\Lambda(s)$ is a Lagrange multiplier, see [40]. The Lagrangian function coincides with the total elastic energy over solutions of the corresponding Euler-Lagrange equations. The internal bending moment is directly related to the curvature $\kappa(s)$.

The continuous action functional $\mathcal{S}$ is defined as:

$$\mathcal{S}\left[\mathbf{q}\right]=\int_0^L\widetilde{\mathcal{L}}\left(\mathbf{q},\mathbf{q}',\mathbf{q}'',\Lambda\right)ds. \tag{5.2.4}$$

Applying Hamilton's principle of stationary action, $\delta\mathcal{S}=0$, yields the Euler-Lagrange equations

$$\begin{gathered}\frac{d^2}{ds^2}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{q}''}\right)-\frac{d}{ds}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{q}'}\right)+\frac{\partial\mathcal{L}}{\partial\mathbf{q}}=\frac{d}{ds}\left(\frac{\partial\Phi}{\partial\mathbf{q}'}\Lambda\right)-\frac{\partial\Phi}{\partial\mathbf{q}}\Lambda,\\ \left\|\mathbf{q}'\right\|^2-1=0,\end{gathered} \tag{5.2.5}$$

which need to be satisfied together with the boundary conditions on positions and tangents, i.e.,$\left(\mathbf{q}(0),\mathbf{q}'(0)\right)=\left(\mathbf{q}_0,\mathbf{q}_0'\right)$ and $\left(\mathbf{q}(L),\mathbf{q}'(L)\right)=\left(\mathbf{q}_N,\mathbf{q}_N'\right)$.

5.2.1 Space discretisation of the elastica

The continuous augmented Lagrangian $\widetilde{\mathcal{L}}$ in Equation (5.2.3) and the action integral $\mathcal{S}$ in Equation (5.2.4) are discretised over the beam length L with constant space steps h and $N+1$ equidistant nodes $0=s_0<s_1<\ldots<s_{N-1}<s_N=L$. In second-order systems, the discrete Lagrangian is a function $\widetilde{L}_d: TQ\times TQ\times\mathbb{R}\times\mathbb{R}\to\mathbb{R}$. In this study, we refer to a discretisation of the Lagrangian function proposed in [11] based on the trapezoidal rule:

$$\begin{aligned}&\widetilde{\mathcal{L}}_d\left(\mathbf{q}_k,\mathbf{q}_k',\mathbf{q}_{k+1},\mathbf{q}_{k+1}',\Lambda_k,\Lambda_{k+1}\right)\\&=\frac{h}{2}\left[\widetilde{\mathcal{L}}\left(\mathbf{q}_k,\mathbf{q}_k',\left(\mathbf{q}_k''\right)^-,\Lambda_k\right)+\widetilde{\mathcal{L}}\left(\mathbf{q}_{k+1},\mathbf{q}_{k+1}',\left(\mathbf{q}_{k+1}''\right)^+,\Lambda_{k+1}\right)\right]\end{aligned}$$

where $\mathbf{q}_k$, $\mathbf{q}_k'$ and Λ_k are approximations of $\mathbf{q}\left(s_k\right)$, $\mathbf{q}'\left(s_k\right)$, and $\Lambda\left(s_k\right)$, and the curvature on the interval $\left[s_k,s_{k+1}\right]$ is approximated in terms of lower order derivatives as follows

$$\mathbf{q}''\left(s_k\right)\approx\left(\mathbf{q}_k''\right)^-=\frac{\left(-2\mathbf{q}_{k+1}'-4\mathbf{q}_k'\right)h+6\left(\mathbf{q}_{k+1}-\mathbf{q}_k\right)}{h^2},$$

$$\mathbf{q}''\left(s_{k+1}\right)\approx\left(\mathbf{q}_{k+1}''\right)^+=\frac{\left(4\mathbf{q}_{k+1}'+2\mathbf{q}_k'\right)h-6\left(\mathbf{q}_{k+1}-\mathbf{q}_k\right)}{h^2}.$$

This amounts to a piece-wise linear and discontinuous approximation of the curvature on $[0, L]$.

The action integral in Equation (5.2.4) along the exact solution $\mathbf{q}$ with boundary conditions $\left(\mathbf{q}_0, \mathbf{q}_0'\right)$ and $\left(\mathbf{q}_N, \mathbf{q}_N'\right)$ is approximated by

$$\mathcal{S}_d = \sum_{k=0}^{N-1} \widetilde{\mathcal{L}}_d \left(\mathbf{q}_k, \mathbf{q}_k', \mathbf{q}_{k+1}, \mathbf{q}_{k+1}', \Lambda_k, \Lambda_{k+1}\right). \tag{5.2.6}$$

The discrete variational principle $\delta \mathcal{S}_d = 0$ leads to the following discrete Euler-Lagrange equations:

$$\begin{aligned}
&D_3\widetilde{\mathcal{L}}_d \left(\mathbf{q}_{k-1}, \mathbf{q}_{k-1}', \mathbf{q}_k, \mathbf{q}_k', \Lambda_{k-1}, \Lambda_k\right) + D_1\widetilde{\mathcal{L}}_d \left(\mathbf{q}_k, \mathbf{q}_k', \mathbf{q}_{k+1}, \mathbf{q}_{k+1}', \Lambda_k, \Lambda_{k+1}\right) = 0,\\
&D_4\widetilde{\mathcal{L}}_d \left(\mathbf{q}_{k-1}, \mathbf{q}_{k-1}', \mathbf{q}_k, \mathbf{q}_k', \Lambda_{k-1}, \Lambda_k\right) + D_2\widetilde{\mathcal{L}}_d \left(\mathbf{q}_k, \mathbf{q}_k', \mathbf{q}_{k+1}, \mathbf{q}_{k+1}', \Lambda_k, \Lambda_{k+1}\right) = 0,\\
&D_6\widetilde{\mathcal{L}}_d \left(\mathbf{q}_{k-1}, \mathbf{q}_{k-1}', \mathbf{q}_k, \mathbf{q}_k', \Lambda_{k-1}, \Lambda_k\right) + D_5\widetilde{\mathcal{L}}_d \left(\mathbf{q}_k, \mathbf{q}_k', \mathbf{q}_{k+1}, \mathbf{q}_{k+1}', \Lambda_k, \Lambda_{k+1}\right) = 0,
\end{aligned} \tag{5.2.7}$$

for $k = 1, \ldots, N-1$, which approximate the equilibrium equations of the beam in Equations (5.2.5) and can be solved together with the boundary conditions.

5.2.2 Data generation

The elastica was one of the first examples displaying elastic instability and bifurcation phenomena [2,42]. Elastic instability implies that small perturbations of the boundary conditions might lead to large changes in the beam configuration, which results in unstable equilibria. Under certain boundary conditions, bifurcation can appear leading to a multiplicity of solutions [27]. In particular, this means that the numerical problem may display history-dependence and converge to solutions that do not minimise the bending energy. In order to generate a physically meaningful data set, avoiding unstable and non-unique solutions is essential. Thus, in addition to the minimisation of the discrete action S_d in Equation (5.2.6), we ensure the fulfilment of the discrete Euler-Lagrange equations (5.2.7), which can be seen as necessary conditions for the stationarity of the discrete action. We exclude from the data set numerical solutions computed with boundary conditions where minimisation of Equation (5.2.6) and accurate solution of Equations (5.2.7) can not be simultaneously achieved.

In particular, we consider a curve of length $L = 3.3$ and bending stiffness $EI = 10$, divided into $N = 50$ intervals. We fix the endpoints $\mathbf{q}_0 = (0,0)$, $\mathbf{q}_N = (3,0)$. The units of measurement are deliberately omitted as they have no impact on the results of this work. We impose boundary conditions on the tangents in the following two variants:

1. the angle of the tangents with respect to the x-axis at the boundary, θ_0 and θ_N, is prescribed in the range $[0, 2\pi]$, in a specular symmetric fashion, i.e.,$\theta_N = \pi - \theta_0$. Hereafter, we refer to this case as *both-ends*,

2. the angle of the left tangent is left fixed as $\theta_0 = 0$ and the angle of the right tangent, θ_N, varies in the range of $[0, 2\pi]$. We refer to this case as *right-end*.

Based on these parameters and boundary values, we generate a data set of 2000 trajectories (1000 trajectories for each case) by minimising the particular action in Equation (5.2.6), with the `trust-constr` solver of the `optimize.minimize` procedure provided in `SciPy` [43]. We check the resulting solutions by using them as initial guesses for the `optimize.root` method of `SciPy`, solving the discrete Euler-Lagrange equations (5.2.7).

5.3 Approximation with neural networks

We start providing a concise overview on neural networks, and we refer to [15,18] and references therein for a more comprehensive introduction. A neural network is a parametric function $f_{\boldsymbol{\rho}} : \mathcal{I} \to \mathcal{O}$ with parameters $\boldsymbol{\rho} \in \Psi$ given as a composition of multiple transformations,

$$f_{\boldsymbol{\rho}} := f_\ell \circ \cdots \circ f_j \circ \cdots \circ f_1, \tag{5.3.1}$$

where each f_j represents the j-th layer of the network, with $j = 1, \dots, \ell$, and ℓ is the number of layers. For example, multi-layer perceptrons (MLPs) have each layer f_j defined as

$$f_j^{MLP}(\mathbf{x}) = \sigma\left(\mathbf{A}_j \mathbf{x} + \mathbf{b}_j\right) \in \mathbb{R}^{n_j}, \tag{5.3.2}$$

where $\mathbf{x} \in \mathbb{R}^{n_{j-1}}$, and $\mathbf{A}_j \in \mathbb{R}^{n_j \times n_{j-1}}$, $\mathbf{b}_j \in \mathbb{R}^{n_j}$ are the parameters of the j-th layer, i.e., $\boldsymbol{\rho} = \left\{\mathbf{A}_j, \mathbf{b}_j\right\}_{j=1}^{\ell}$. The activation function σ is a continuous nonlinear scalar function, which acts component-wise on vectors. The architecture of the neural network is prescribed by the layers f_j in Equation (5.3.1) and determines the space of functions $\mathcal{F} = \left\{f_{\boldsymbol{\rho}} : \mathcal{I} \to \mathcal{O},\ \boldsymbol{\rho} \in \Psi\right\}$ that can be represented. The weights $\boldsymbol{\rho}$ are chosen such that $f_{\boldsymbol{\rho}}$ approximates accurately enough a map of interest $f : \mathcal{I} \to \mathcal{O}$. Usually, this choice follows from minimising a purposely designed loss function $\mathrm{Loss}(\boldsymbol{\rho})$.

In supervised learning, we are given a data set $\Omega = \left\{\mathbf{x}^i, \mathbf{y}^i\right\}_{i=1}^{M}$ consisting of M pairs $\left(\mathbf{x}^i, \mathbf{y}^i = f\left(\mathbf{x}^i\right)\right)$. The loss function is measuring the distance between

the network predictions $f_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)$ and the desired outputs $\mathbf{y}^i$ in some appropriate norm $\|\cdot\|$

$$\text{Loss}(\boldsymbol{\rho}) = \frac{1}{M}\sum_{i=1}^{M}\left\| f_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right) - \mathbf{y}^i\right\|^2 .$$

The training of the network is the process of minimising $\text{Loss}(\boldsymbol{\rho})$ with respect to $\boldsymbol{\rho}$ and it is usually done with gradient descent (GD):

$$\boldsymbol{\rho}^{(k)} \mapsto \boldsymbol{\rho}^{(k)} - \eta\nabla\text{Loss}\left(\boldsymbol{\rho}^{(k)}\right) =: \boldsymbol{\rho}^{(k+1)}.$$

The scalar value η is known as the learning rate. The iteration process is often implemented using subsets of data $\mathcal{B} \subset \Omega$ of cardinality $B = |\mathcal{B}|$ (batches). In this paper we use an accelerated version of GD known as Adam [17].

Once the training is complete, we assess the model's accuracy in predicting the correct output for new inputs included in the test set that are unseen during training. In the following, we measure the accuracy on both the training and the test data using the mean squared error of the difference between the predicted trajectories and the true ones.

We now turn to the task of approximating the static equilibria of the planar elastica introduced in Section 5.2, i.e., approximating a family of curves $\left\{\mathbf{q}^i : [0,L] \mapsto \mathbb{R}^2\right\}$ determined by boundary conditions,

$$\left\{\mathbf{q}^i(0) = \mathbf{q}_0^i,\, \mathbf{q}^i(L) = \mathbf{q}_N^i,\, \left(\mathbf{q}^i\right)'(0) = \left(\mathbf{q}_0^i\right)',\, \left(\mathbf{q}^i\right)'(L) = \left(\mathbf{q}_N^i\right)'\right\}, \tag{5.3.3}$$

where $\left(\mathbf{q}_0^i, \mathbf{q}_N^i, \left(\mathbf{q}_0^i\right)', \left(\mathbf{q}_N^i\right)'\right) \in \mathbb{R}^8$. In order to tackle this problem, we require a set of evaluations $\left\{\mathbf{q}_k^i, \left(\mathbf{q}_k^i\right)'\right\}$ on the nodes $s_k \in [0,L]$ of a discretisation. More precisely, in our setting, the data set includes numerical approximations $\hat{\mathbf{q}}$ of the solution $\mathbf{q}(s)$ and its spatial derivative $\mathbf{q}'(s)$ at the $N-1$ discrete locations $s_k = \frac{kh}{L}$ in the interval $[0,L]$, for M sets of boundary conditions, as described in Section 5.2.2.

5.4 The discrete network

The discretisation of Euler's elastica presented in Section 5.2.1 provides discrete solutions on a set of nodes along the curve. These solutions can sometimes be hard to obtain since a non-convex optimisation problem needs to be solved, and the number of nodes can be large. This motivates the use of neural networks to learn the approximate solution on the internal nodes, for a given set of boundary conditions. The data set Ω consists of M precomputed discrete

solutions

$$\Omega = \left\{ \left(\mathbf{x}^i, \mathbf{y}^i \right) \right\}_{i=1}^{M},$$

where

$$\mathbf{x}^i = \left(\mathbf{q}_0^i, \mathbf{q}_N^i, \left(\mathbf{q}_0^i \right)', \left(\mathbf{q}_N^i \right)' \right) \in \mathbb{R}^8$$

are the input boundary conditions and

$$\mathbf{y}^i = \left(\hat{\mathbf{q}}_1^i, \ldots, \hat{\mathbf{q}}_{N-1}^i, \left(\hat{\mathbf{q}}_1^i \right)', \ldots, \left(\hat{\mathbf{q}}_{N-1}^i \right)' \right) \in \mathbb{R}^{4(N-1)}$$

are the computed solutions at the internal nodes that serve as output data for the training of the network.

For any symmetric positive definite matrix W, we define the weighted norm $\|\mathbf{x}\|_W = \|W\mathbf{x}\|_2$. The weighted MSE loss

$$\text{Loss}(\boldsymbol{\rho}) = \frac{1}{4M(N-1)} \sum_{i=1}^{M} \left\| q_{\boldsymbol{\rho}}^{\text{d}} \left(\mathbf{x}^i \right) - \mathbf{y}^i \right\|_W^2, \tag{5.4.1}$$

will be used to learn the input-to output map $q_{\boldsymbol{\rho}}^{\text{d}} : \mathbb{R}^8 \to \mathbb{R}^{4(N-1)}$, where the superscript d stands for discrete. One should be aware that there is a numerical error in $\mathbf{y}^i$ compared to the exact solution and the size of this error will pose a limit to the accuracy of the neural network approximation.

5.4.1 Numerical experiments

This section provides experimental support to the proposed learning framework. We perform a series of experiments varying the architecture of the neural network and the hyperparameters in the training procedure. The codes to run the experiments in this work are written using the machine learning library `PyTorch` [32]. We use the Adam optimiser [17] for the training, carefully selecting learning rate and weight decay to prevent over-fitting, see Table 5.2. In (5.4.1) we use the weight matrix

$$W = I + \gamma G^T G$$

where $G = S - I$ with S the forward shift operator on vectors of $\mathbb{R}^{4(N-1)}$. We test a range of different batch sizes $B \leq M$ and fix the total number of epochs to 300. Finally, we also test for the influence of performing normalisation of the input data. We collect in Table 5.2 all the hyperparameters and network architectures with their corresponding ranges.

We rely on the software framework `Optuna` [1] to automate and efficiently conduct the search for the combination that yields the best result. This is

Hyperparameter	Range	Distribution
architecture	{MLP, ResNet}	discrete uniform
normalisation	{True, False}	discrete uniform
activation function σ	{Tanh, Swish, Sigmoid, ReLU, LeakyReLU}	discrete uniform
#layers ℓ	$\{0,1,2,3,4\}$	discrete uniform
#hidden nodes in each layer	$[32,1024]\cap\mathbb{N}$	discrete uniform
learning rate η	$[1\cdot10^{-4},1\cdot10^{-1}]$	log uniform
weight decay	$[1\cdot10^{-7},5\cdot10^{-4}]$	log uniform
γ	$[0,1\cdot10^{-2}]$	uniform
batch size	$\{32,64,128\}$	discrete uniform

Table 5.2: Hyperparameter ranges for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set. The first column of the table reports the hyperparameters and network architectures we test for. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`. MLP corresponds to the network in Equation (5.3.2), Section 5.3, while ResNet is a residual neural network defined in Equation (5.A.1) of Appendix 5.A.

reported in Table 5.3. The resulting training error on the *both-end* data set is $2.791\cdot10^{-7}$, and the test error on a set of trajectories belonging to the same data set is $3.028\cdot10^{-7}$. Figure 5.1 compares test trajectories for $\mathbf{q}$ and $\mathbf{q}'$. We remark that, as already clear from the low value of the training and test errors, the network can accurately replicate the behaviour of the training and test data. Furthermore, since the network is trained only on the internal nodes and the boundary values are appended to the predicted solution in a post processing phase, we have zero errors at the end nodes. On the other hand, since this discrete approach does not relate the components as evaluations of a smooth curve, there is no regular behaviour in the error.

As an additional evaluation of the deep learning framework's behaviour, we conduct experiments to assess how the learning process performs when the number of training data varies, i.e., with different splittings of the data set into training and test sets. We report the results in Table 5.4 and summarise the corresponding hyperparameters in Table 5.12 of the Appendix.

We also report results obtained by merging the *both-end* and the *right-end* trajectories, with 90%−10% splitting of the whole new data set into training and test set. The results are shown in Figure 5.2 and the selected hyperparameters are collected in Table 5.5. The resulting training and test errors are, respectively, $3.047\cdot10^{-7}$ and $3.141\cdot10^{-7}$. Finally, we remark that the test accuracy is a good measure of the generalisation error of neural network under the hypothesis that

Selected hyperparameters	
architecture	MLP
normalisation	True
activation function σ	Tanh
#layers ℓ	4
#hidden nodes in each layer	879
learning rate η	$1.378 \cdot 10^{-3}$
weight decay	$1.535 \cdot 10^{-7}$
γ	$4.242 \cdot 10^{-3}$
batch size	32

Table 5.3: Combination of hyperparameters yielding the best results for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90% – 10% splitting into training and test set. The results are shown in Figure 5.1.

Data set splitting Training - test	Training accuracy	Test accuracy
10% - 10%	$6.530 \cdot 10^{-5}$	$5.636 \cdot 10^{-4}$
20% - 10%	$2.096 \cdot 10^{-5}$	$3.457 \cdot 10^{-5}$
40% - 10%	$2.186 \cdot 10^{-6}$	$3.494 \cdot 10^{-6}$
90% - 10%	$2.791 \cdot 10^{-7}$	$3.028 \cdot 10^{-7}$

Table 5.4: Behaviour of the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with fewer training data points. The size of the training set varies, while that of the test set is fixed. The last row corresponds to the results in Figure 5.1.

the test and the training sets are independent of each other, but follow the same distribution. If we test over input boundary conditions that not only are unseen during the training, but also do not belong in the the same range of the training data, the resulting accuracy is expected to be low, since neural networks are in general not able to perform this sort of extrapolation. To show this, we consider the neural network trained and tested over the *both-ends* data set, related to the results in Figure 5.1 and in the last row of Table 5.4. We use 10% of the *right-end* data set as a test set, and we obtain a test error equal to $2.228 \cdot 10^{-2}$. This highlights that care must be taken when using the trained network to make inference over new input data.

5.5 The continuous network

The approach described in the previous section shows accurate results, given a large enough amount of beam discretisations with a fixed number of nodes $N+1$,

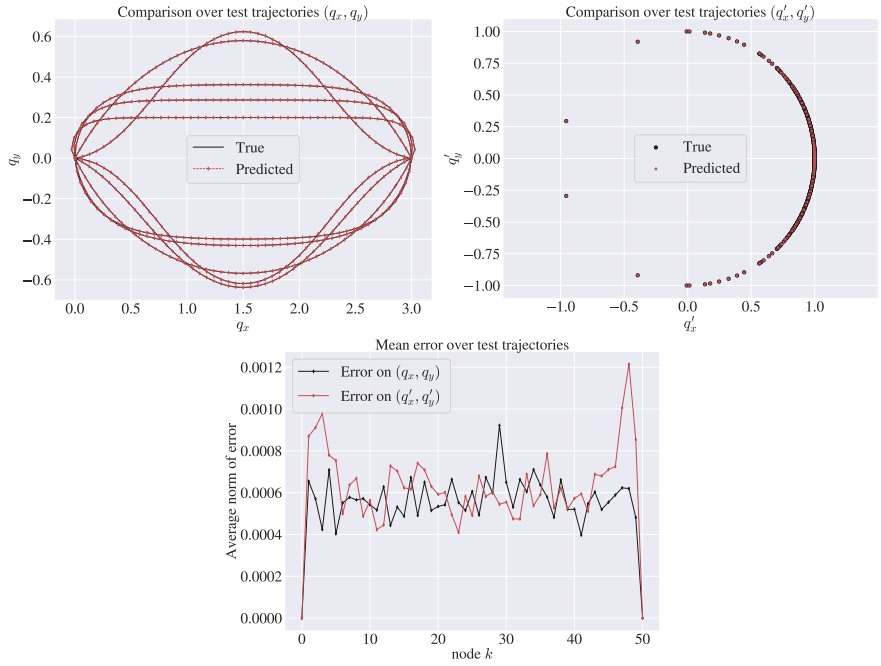


Figure 5.1: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90% – 10% splitting into training and test set. These results are obtained with the hyperparameters from Table 5.3, that yield a training error equal to $2.791 \cdot 10^{-7}$ and a test error equal to $3.028 \cdot 10^{-7}$. For presentation purposes, only 10 randomly selected trajectories are considered in the first two plots.

equally distributed in $[0, L]$. It seems reasonable to expect that the parametric model's approximation quality improves when the number of discretisation nodes increases. However, in this approach, the dimension of the predicted vector grows with N, and hence minimising the loss function (5.4.1) becomes more difficult. In addition, the fact that the discrete network approach depends on the spatial discretisation of the training data restricts the output dimension to a specific number of nodes. Consequently, there would be two main options to assess the solution at different locations: training the network once more, or interpolating the previously obtained approximation. These limitations make such a discrete approach less appealing and suggest that having a neural network that is a smooth function of the arc length coordinate s can be beneficial. This modelling assumption would also be compatible with different discretisations of the curve and would not suffer from the curse of dimensionality if more nodes were added. In this setting, the discrete node s_k at which an approximation of

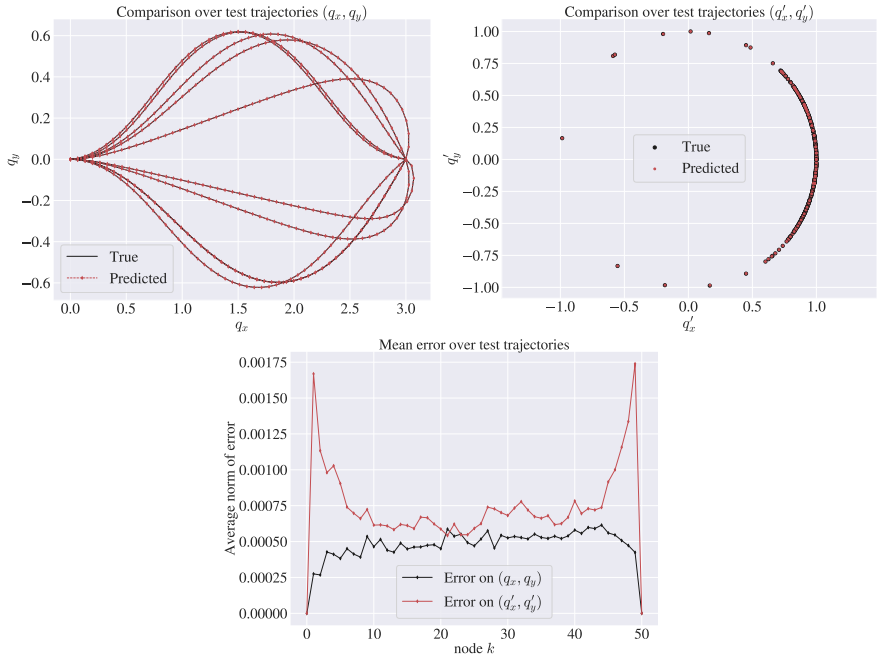


Figure 5.2: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* + *right-end* data set with 90% – 10% splitting into training and test set. These results are obtained with the hyperparameters from Table 5.5, that yield a training error equal to $3.047 \cdot 10^{-7}$ and a test error equal to $3.141 \cdot 10^{-7}$. For presentation purposes, only 10 randomly selected trajectories are considered in the first two plots.

the solution is available, is included in the input data together with the boundary conditions. As a result, we work with the following data set

$$\Omega = \left\{ \left(s_k, \mathbf{x}^i \right), \mathbf{y}^i_k \right\}^{i=1,\dots,M}_{k=0,\dots,N},$$

where, as in the previous section,

$$\mathbf{x}^i = \left(\mathbf{q}^i_0, \mathbf{q}^i_N, \left(\mathbf{q}^i_0 \right)', \left(\mathbf{q}^i_N \right)' \right) \in \mathbb{R}^8,$$

and

$$\mathbf{y}^i_k = \left(\hat{\mathbf{q}}^i_k, \left(\hat{\mathbf{q}}^i_k \right)' \right).$$

Here $\hat{\mathbf{q}}^i_k$ is the numerical solution $\hat{\mathbf{q}}$ on the node s_k, satisfying the i-th boundary conditions in Equation (5.3.3). Let us introduce the neural network

$$q^{\mathrm{c}}_{\boldsymbol{\rho}} : \mathbb{R}^8 \to \mathcal{C}^\infty \left([0, L], \mathbb{R}^2 \right),$$

Selected hyperparameters	
architecture	MLP
normalisation	True
activation function σ	LeakyReLU
#layers ℓ	2
#hidden nodes in each layer	1006
learning rate η	$3.611 \cdot 10^{-3}$
weight decay	$1.515 \cdot 10^{-7}$
γ	$6.388 \cdot 10^{-3}$
batch size	32

Table 5.5: Combination of hyperparameters yielding the best results for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends + right-end* data set with 90%-− 10% splitting into training and test set. The results are shown in Figure 5.2.

and the differential operator

$$\mathcal{D}: \mathcal{C}^\infty\left([0,L],\mathbb{R}^2\right) \to \mathcal{C}^\infty\left([0,L],\mathbb{R}^2\right),\ \mathcal{D}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)\right)(s_k) = \frac{d}{ds}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)\right)(s)\Big|_{s=s_k},$$

so that we can define

$$y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) := \left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k), \mathcal{D}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)\right)(s_k)\right).$$

To train the network $q^c_{\boldsymbol{\rho}}$, we define the loss function

$$\begin{aligned}\mathrm{Loss}(\boldsymbol{\rho}) = \frac{1}{4M(N+1)} \sum_{i=1}^{M}\sum_{k=0}^{N}\Bigg(&\left\| y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) - \boldsymbol{y}^i_k\right\|_2^2 \\ &+ \gamma\left(\left\|\pi_{\mathcal{D}}\left(y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k)\right)\right\|_2^2 - 1\right)^2\Bigg),\end{aligned} \tag{5.5.1}$$

where $\pi_{\mathcal{D}}: \mathbb{R}^8 \to \mathbb{R}^4$ is the projection on the second component $\mathcal{D}\left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)\right)(s_k)$, and $\gamma \geq 0$ weighs the violation of the normality constraint. The map $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ is now a neural network that associates each set of boundary conditions $\boldsymbol{x}^i$ with a smooth curve $q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right): [0,L] \to \mathbb{R}^2$ that can be evaluated at every point $s \in [0,L]$. We denote this network with the superscript c, since this curve is in particular continuous. The outputs $q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) \in \mathbb{R}^2$ are approximations of the configuration of the beam at $s \in [0,L]$.

We point out that, contrarily to the discrete case, here we learn approximations of $\mathbf{q}(s)$ also on the end nodes, i.e., at $s = 0$ and $s = L$. This is due to the fact

that we do not impose the boundary conditions by construction. Even though there are multiple approaches to embed them into the network architecture, the one we try in our experiments made the optimisation problem too difficult, thus we only impose the boundary conditions weakly in the loss function.

Another strategy is to compute the angles θ_k between the tangents $(\hat{\mathbf{q}}_k)'$ and the x-axis and to use them as training data. To this end, we define the neural network

$$\theta^{\mathrm{c}}_{\boldsymbol{\rho}} : \mathbb{R}^8 \to \mathcal{C}^\infty\left([0, L], \mathbb{R}\right)$$

as $\theta^{\mathrm{c}}_{\boldsymbol{\rho}} = \hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}} \circ \pi$, where

$$\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}} : \mathbb{R}^2 \to \mathcal{C}^\infty\left([0, L], \mathbb{R}\right) \tag{5.5.2}$$

is a neural network and the function $\pi : \mathbb{R}^8 \to \mathbb{R}^2$ extracts the tangential angles from the boundary conditions, i.e., $\pi\left(\boldsymbol{x}^i\right) = \left(\theta^i_0, \theta^i_N\right)$. Such a network should approximate the angular function $\theta : [0, L] \ni s \to \mathbb{R}$, so that

$$\tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) := \left(\cos\left(\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s)\right), \sin\left(\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s)\right)\right) \in \mathbb{R}^2 \tag{5.5.3}$$

gets close to the tangent vector $\mathbf{q}'(s)$. As a result, the constraint on the unit norm of the tangents is satisfied by construction, and the inextensibility of the elastica is guaranteed. The curve

$$\mathbf{q}(s) = \mathbf{q}_0 + \int_0^s \mathbf{q}'(\bar{s}) \mathrm{d}\bar{s}$$

can then be approximated through the reconstruction formula

$$q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) = \mathbf{q}_0 + \mathcal{I}\left(\tau^{\mathrm{c}}_{\rho}\left(\boldsymbol{x}^i\right)\right)(s), \tag{5.5.4}$$

where the operator $\mathcal{I} : \mathcal{C}^\infty\left([0, L], \mathbb{R}^2\right) \to \mathcal{C}^\infty\left([0, L], \mathbb{R}^2\right)$ is such that

$$\mathcal{I}\left(\tau^{\mathrm{c}}_{\rho}\left(\boldsymbol{x}^i\right)\right)(s) \approx \int_0^s \tau^{\mathrm{c}}_{\rho}\left(\mathbf{x}^i\right)(\bar{s}) \mathrm{d}\bar{s}.$$

In the numerical experiments, $\mathcal{I}$ is based on the 3-point Gaussian quadrature formula applied to a partition of the interval $[0, L]$, see [33, Chapter 9]. As done previously, we define the vector

$$y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) := \left(q^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k), \tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s_k)\right), \tag{5.5.5}$$

with components defined as in Equations (5.5.3) and (5.5.4). This allows us to train the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ by minimising the same loss function as in Equation

(5.5.1), where this time $y^{\mathrm{c}}_{\boldsymbol{\rho}}$ is given by Equation (5.5.5). Furthermore, since by construction this case satisfies $\left\| \pi_{\mathcal{D}} \left(y^{\mathrm{c}}_{\boldsymbol{\rho}} \left(\boldsymbol{x}^i \right) (s) \right) \right\|_2 = \left\| \tau^{\mathrm{c}}_{\boldsymbol{\rho}} \left(\boldsymbol{x}^i \right) (s) \right\|_2 \equiv 1$, we set $\gamma = 0$. We present numerical experiments for the two proposed continuous networks $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ and $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$. In the latter case, by neural network architecture we refer to $\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}$ rather than $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ in what follows. We analyse $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ more thoroughly in Section 5.5.1, mirroring most of the discrete case experiments. In Section 5.5.2 we study how the results are affected when we impose the arc length parametrization and enforce the boundary conditions to be exactly satisfied by the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$.

5.5.1 Numerical experiments with $q^{\mathrm{c}}_{\boldsymbol{\rho}}$

As for the case of the discrete network, we perform an in-depth investigation of this learning setting by varying the architecture of the continuous neural network and the hyperparameters in the training procedure, whose range of options can be found in Table 5.6. In this case, we define the loss as in Equation (5.5.1), with $\gamma = 10^{-2}$. The weight decay is systematically set to 0.

Hyperparameter	Range	Distribution
architecture	{MLP, ResNet, MULT}	discrete uniform
normalisation	{True, False}	discrete uniform
activation function σ	{Tanh, Swish, Sigmoid, Sine}	discrete uniform
#layers ℓ	$\{3, \dots, 8\}$	discrete uniform
#hidden nodes in each layer	$[10, 200] \cap \mathbb{N}$	discrete uniform
learning rate η	$[1 \cdot 10^{-4}, 1 \cdot 10^{-1}]$	log uniform

Table 5.6: Hyperparameter ranges for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90% – 10% splitting into training and test set. The first column of the table reports the hyperparameters and network architectures we test for. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`. The weight decay is systematically set to 0. MULT stands for multiplicative neural network and corresponds to the network in Equations (5.A.2)-(5.A.6) of Appendix 5.A.

Table 5.7 collects the combination of hyperparameters yielding the best results on the *both-ends* data set. This leads to a training error equal to $1.869 \cdot 10^{-6}$ and a test error equal to $4.81 \cdot 10^{-6}$. In Figure 5.3, the comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ is shown. As we can see in the plot showing the mean error over the trajectories, the error on the end nodes is nonzero, since we are not imposing boundary conditions by construction. This is in contrast to the corresponding plot for the discrete network in Figure 5.1.

Selected hyperparameters	
architecture	MULT
normalisation	True
activation function σ	Tanh
#layers ℓ	5
#hidden nodes in each layer	190
learning rate η	$3.025 \cdot 10^{-3}$

Table 5.7: Combination of hyperparameters yielding the best results for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90%−10% splitting into training and test set. The results are shown in Figure 5.3.

Also in this case, we examine the behaviour of the learning process with different splittings of the data set into training and test sets. We display the results in Table 5.8 and summarise the corresponding hyperparameters in Appendix 5.B, Table 5.13.

Data set splitting Training - test	Training accuracy	Test accuracy
10% - 10%	$2.383 \cdot 10^{-4}$	$7.784 \cdot 10^{-4}$
20% - 10%	$5.612 \cdot 10^{-5}$	$7.285 \cdot 10^{-5}$
40% - 10%	$7.104 \cdot 10^{-6}$	$9.275 \cdot 10^{-6}$
90% - 10%	$1.869 \cdot 10^{-6}$	$4.810 \cdot 10^{-6}$

Table 5.8: Behaviour of the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with fewer training data points. The size of the training set varies, while that of the test set is fixed. The last row corresponds to the results in Figure 5.3.

5.5.2 Numerical experiments with $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$

Here we consider a neural network approximation of the angle $\theta(s)$ that parametrises the tangent vector $\mathbf{q}'(s) = \Big(\cos\big(\theta(s)\big), \sin\big(\theta(s)\big)\Big)$. By design, the approximation $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}$ of the tangent vector $\mathbf{q}'$ satisfies the constraint $\left\| \tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) \right\|_2 = 1$ for every $s \in [0, L]$ and $\boldsymbol{x}^i \in \mathbb{R}^8$. We also analyse how the neural network approximation behaves when the boundary conditions $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(0) = \mathbf{q}'(0)$ and $\tau^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(L) = \mathbf{q}'(L)$ are imposed by construction. To do so, we model the parametric function $\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}$, defined in Equation (5.5.2), in one of the two following ways:

$$\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) = f_{\boldsymbol{\rho}}(s, \theta^i_0, \theta^i_N), \tag{5.5.6}$$

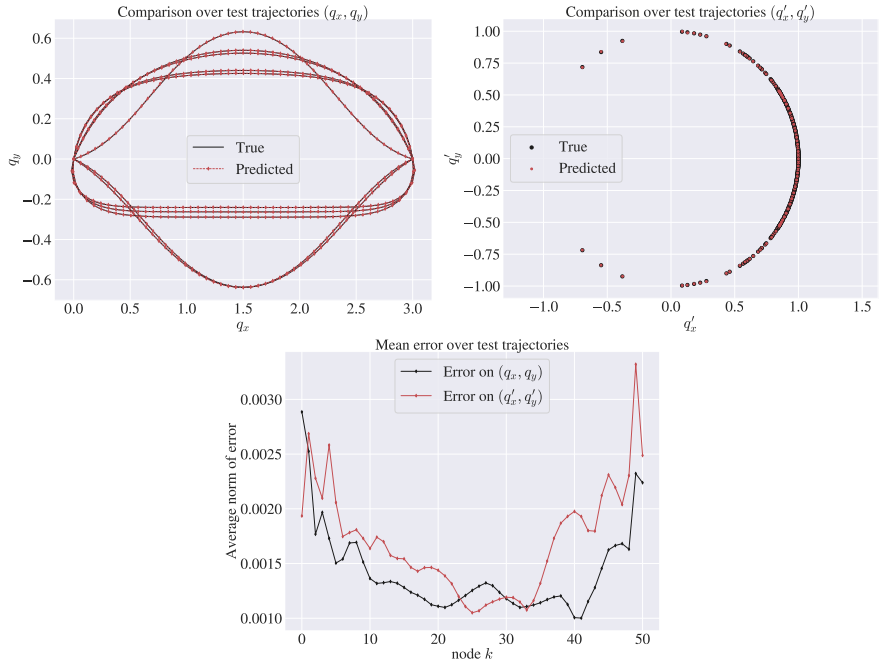


Figure 5.3: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90% – 10% splitting into training and test set. These results are obtained with the hyperparameters from Table 5.7, that yield a training error equal to $1.869 \cdot 10^{-6}$ and a test error equal to $4.810 \cdot 10^{-6}$. For presentation purposes, only 10 randomly selected trajectories are considered in the first two plots.

$$\begin{aligned}\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) = f_{\boldsymbol{\rho}}(s, \theta^i_0, \theta^i_N) + (\theta^i_0 - f_{\boldsymbol{\rho}}(0, \theta^i_0, \theta^i_N))e^{-100s^2} \\ + (\theta^i_N - f_{\boldsymbol{\rho}}(L, \theta^i_0, \theta^i_N))e^{-100(s-L)^2},\end{aligned} \tag{5.5.7}$$

where $f_{\boldsymbol{\rho}} : \mathbb{R}^3 \to \mathbb{R}$ is any neural network, and we recall that $\pi\left(\boldsymbol{x}^i\right) = \left(\theta^i_0, \theta^i_N\right)$. We remark that, in the case of the parameterisation in Equation (5.5.7), one gets $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(0) = \theta^i_0$ and $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(L) = \theta^i_N$ up to machine precision, due to the fast decay of the Gaussian function. As in the previous sections, we collect the hyperparameter and architecture options with the respective range of choices in Table 5.9, and we report the results without imposing the boundary conditions in Figure 5.4, while those imposing them in Figure 5.5, in both cases using the *both-ends* data set, with 90% – 10% splitting into training and test set. The results shown in the two figures correspond respectively to training errors of $5.821 \cdot 10^{-6}$ and $6.068 \cdot 10^{-6}$, and test errors of $6.231 \cdot 10^{-6}$ and $6.289 \cdot 10^{-6}$. The best performing hyperparameter combinations can be found in Tables 5.10 and

5.11.

Hyperparameter	Range	Distribution
architecture	{MLP, ResNet, MULT}	discrete uniform
activation function σ	{Tanh, Swish, Sigmoid}	discrete uniform
#layers ℓ	$\{3, \dots, 8\}$	discrete uniform
#hidden nodes in each layer	$[50, 200] \cap \mathbb{N}$	discrete uniform
learning rate η	$[1 \cdot 10^{-3}, 1 \cdot 10^{-1}]$	log uniform

Table 5.9: Hyperparameter ranges for the continuous network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set. The first column of the table reports the hyperparameters and network architectures we test for. The second describes the set of allowed values for each, while the third specifies how such values are explored through `Optuna`. The weight decay is systematically set to 0, and no normalisation is applied.

Selected hyperparameters	
architecture	MULT
activation function σ	Tanh
#layers ℓ	7
#hidden nodes in each layer	143
learning rate η	$3.007 \cdot 10^{-3}$

Table 5.10: Combination of hyperparameters yielding the best results for the case when $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in Equation (5.5.6), with 90% – 10% splitting of the *both-ends* data set into training and test set. The results are shown in Figure 5.4.

5.6 Discussion

The results in Figures 5.4 and 5.5 are comparable, especially looking at the mean error plots. This suggests that the imposition of the boundary conditions, in the proposed way, is not limiting the expressivity of the considered network. Thus, given the boundary value nature of our problem, these figures advocate the enforcement of the boundary conditions on the network $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$. However, due to the chosen reconstruction procedure in Equation (5.5.4) for the variable $\mathbf{q}$, we are able to impose the boundary conditions on $\mathbf{q}$ only on the left node. Clearly, other more symmetric reconstruction procedures can be adopted, but the proposed one has proved to provide better experimental results.

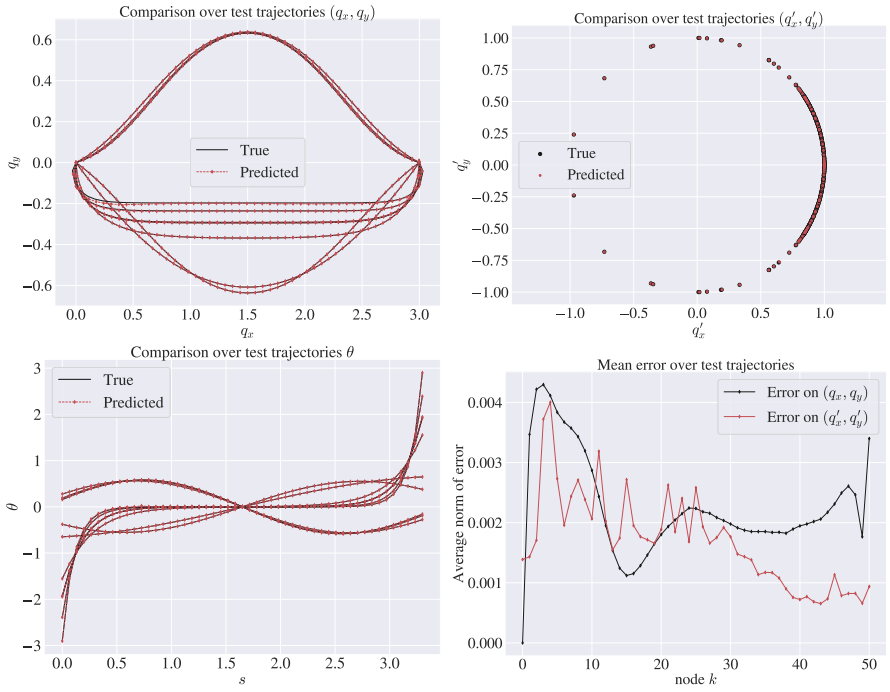


Figure 5.4: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in Equation (5.5.6), with 90% − 10% splitting of the *both-ends* data set into training and test set. These results are obtained with the hyperparameters from Table 5.10, that yield a training error equal to $5.821 \cdot 10^{-6}$ and a test error equal to $6.231 \cdot 10^{-6}$. For presentation purposes, only 10 randomly selected trajectories are considered in the first two plots.

Comparing the results related to $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ with those of $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$, we notice similar performances in terms of training and test errors. In both the cases, they have one order of magnitude more than the corresponding training and test errors of the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$. Thus, as a results of our experiments, we can conclude that

- if the accuracy and the efficient evaluation of the model at the discrete nodes are of interest, the discrete network is the best option;
- for a more flexible model, not restricted to the discrete nodes, the continuous network is a better choice; among the two proposed modelling strategies, working with $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ is more suitable for an easy parametrisation of both $\mathbf{q}$ and $\mathbf{q}'$, while $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is more suitable to impose geometrical structure and constraints.

The total accuracy error of a neural network model can be defined by splitting

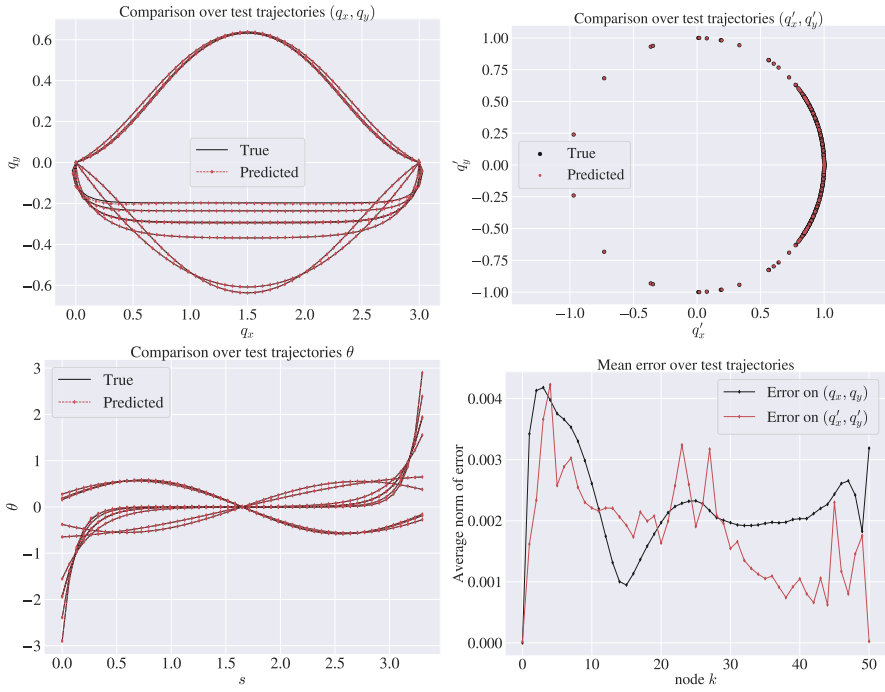


Figure 5.5: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in Equation (5.5.7), with 90% − 10% splitting of the *both-ends* data set into training and test set. These results are obtained with the hyperparameters from Table 5.11, that yield a training error equal to $6.068 \cdot 10^{-6}$ and a test error equal to $6.289 \cdot 10^{-6}$. For presentation purposes, only 10 randomly selected trajectories are considered in the first two plots.

it into three components: approximation error, optimisation error, and generalisation error (see, e.g., [23]). To achieve excellent agreement between predicted and reference trajectories, it is important to select the appropriate architecture and fine-tune the model hyperparameters. Our results demonstrate that we can construct a network that is expressive enough to provide a small approximation error and with very good generalisation capability.

5.6.1 Future work

In the methods presented in this paper, there is no interaction between the mathematical problem and the neural network model once the data set is created. As a way to improve the results presented here, one could include the Euler elastica model directly into the training process. This could be done either by directly imposing in the loss function that $\mathbf{q}(s)$ satisfies the differential equa-

Selected hyperparameters	
architecture	MULT
activation function σ	Tanh
#layers ℓ	4
#hidden nodes in each layer	175
learning rate η	$1.846 \cdot 10^{-3}$

Table 5.11: Combination of hyperparameters yielding the best results for the case when $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in Equation (5.5.7), with 90% – 10% splitting of the *both-ends* data set into training and test set. The results are shown in Figure 5.5.

tions (5.2.5), or one could add the constrained action integral from Equation (5.2.4) into the loss function that is minimised, see e.g. [19, 34, 38, 46].

There are many promising directions in order to follow up this work. One is to consider 3D versions of the Euler elastica, another is to look at the dynamical problem, and finally one may examine industrial applications. As an example, we refer to the modelling of endoscopes due to the high bending deformation of these medical devices under certain loading cases [41]. The approximation of the elastica through neural networks can indeed help in the prediction of the deformed configuration of the beam in constrained narrow environments.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860124. This work was partially supported by a grant from the Simons Foundation (DM). This contribution reflects only the authors' view, and the Research Executive Agency and the European Commission are not responsible for any use that may be made of the information it contains.

Appendix 5.A Other neural network architectures

Another example of neural network architecture, besides the MLP defined in Equation (5.3.2), is the residual neural network (ResNet), where

$$f_j^{RES}(\mathbf{x}) = \mathbf{x} + \mathbf{B}_j^T \sigma(\mathbf{A}_j \mathbf{x} + \mathbf{b}_j) \in \mathbb{R}^n, \tag{5.A.1}$$

with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A}_j, \mathbf{B}_j \in \mathbb{R}^{n_j \times n}$, $\mathbf{b}_j \in \mathbb{R}^{n_j}$, and $\boldsymbol{\rho} = \left\{\mathbf{A}_j, \mathbf{b}_j, \mathbf{B}_j\right\}_{j=1}^{\ell}$. We also provide the expression of the forward propagation of the multiplicative network used

for the experiments in Section 5.5:

$$\mathbf{U} = \sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right), \quad \mathbf{V} = \phi\left(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2\right) \tag{5.A.2}$$

$$\mathbf{H}_1 = \sigma\left(\mathbf{W}_3\mathbf{x} + \mathbf{b}_3\right) \tag{5.A.3}$$

$$\mathbf{Z}_j = \sigma\left(\mathbf{W}_j^z\mathbf{H}_j + \mathbf{b}_j^z\right),\ j = 1,\dots,\ell \tag{5.A.4}$$

$$\mathbf{H}_{j+1} = \left(1 - \mathbf{Z}_j\right) \odot \mathbf{U} + \mathbf{Z}_j \odot \mathbf{V},\ j = 1,\dots,\ell \tag{5.A.5}$$

$$f_{\boldsymbol{\rho}}^{MULT}(\mathbf{x}) = \mathbf{W}\mathbf{H}_{\ell+1} + \mathbf{b}, \tag{5.A.6}$$

where $\odot$ denotes the component-wise multiplications. In this case, $\boldsymbol{\rho} = \left\{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_3, \mathbf{b}_3, \left(\mathbf{W}_j^z, \mathbf{b}_j^z\right)_{j=1}^{\ell}, \mathbf{W}, \mathbf{b}\right\}$, and the weight matrices and biases have shapes that allow for the expressions (5.A.2)-(5.A.6) to be well-defined. This architecture is inspired by neural attention mechanisms and was introduced in [45] to improve the gradient behaviour. A further motivation for our choice of including this architecture is experimental since it has proven effective in solving the task of interest, while still having a similar number of parameters to the MLP architecture. Throughout the paper, we refer to this architecture as *multiplicative* since it includes component-wise multiplications, which help capture multiplicative interactions between the variables.

Appendix 5.B Further results on the hyperparameters of the neural networks

Hyperparameter combination	Data set splitting			
	10% - 10%	20% - 10%	40% - 10%	90% - 10%
architecture	MLP	MLP	MLP	MLP
normalization	False	False	False	True
activation function σ	Tanh	Tanh	Tanh	Tanh
number of layers ℓ	4	4	3	4
#hidden nodes in each layer	950	351	904	879
learning rate η	$1.019 \cdot 10^{-3}$	$5.455 \cdot 10^{-3}$	$1.429 \cdot 10^{-3}$	$1.378 \cdot 10^{-3}$
weight decay	$1.79 \cdot 10^{-6}$	$2.142 \cdot 10^{-7}$	$1.317 \cdot 10^{-7}$	$1.535 \cdot 10^{-7}$
γ	$8.595 \cdot 10^{-3}$	$2.973 \cdot 10^{-3}$	$9.338 \cdot 10^{-3}$	$4.242 \cdot 10^{-3}$
batch size	64	32	32	32

Table 5.12: Choice of hyperparameters for the training of the discrete network $q_{\boldsymbol{\rho}}^{\mathrm{d}}$ tested on the *both-ends* data set with different numbers of training data points.

Hyperparameter combination	Data set splitting			
	10% - 10%	20% - 10%	40% - 10%	90% - 10%
architecture	MULT	MULT	MULT	MULT
normalization	True	True	True	True
activation function σ	Tanh	Tanh	Sine	Tanh
number of layers ℓ	5	6	6	5
#hidden nodes in each layer	193	121	169	190
learning rate η	$4.548 \cdot 10^{-3}$	$4.913 \cdot 10^{-3}$	$4.2 \cdot 10^{-3}$	$3.025 \cdot 10^{-3}$

Table 5.13: Choice of hyperparameters for the training of the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with different numbers of training data points. The weight decay is systematically set to 0.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019. 119

[2] D. Bigoni. *Nonlinear solid mechanics: bifurcation theory and material instability*. Cambridge University Press, (2012). 116

[3] S. C. Brenner. *The mathematical theory of finite element methods*. Springer, (2008). 111

[4] S. L. Brunton and J. N. Kutz. Machine learning for partial differential equations. *arXiv:2303.17078*, 2023. 111

[5] S. Chevalier, J. Stiasny, and S. Chatzivasileiadis. Accelerating dynamical system simulations with contracting and physics-projected neural-newton solvers. In *Learning for Dynamics and Control Conference*, pages 803–816, PMLR, 2022. 111, 112

[6] L. Colombo, S. Ferraro, and D. M. de Diego. Geometric integrators for higher-order variational systems and their application to optimal control. *Journal of Nonlinear Science*, 26:1615–1650, 2016. 113

[7] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022. 111, 112

[8] M. De Florio, E. Schiassi, and R. Furfaro. Physics-informed neural networks and functional interpolation for stiff chemical kinetics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(6), 2022. 111

[9] L. Euler. *De curvis elastici*. Additamentum in Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes, sive solutio problematis isoperimetrici lattissimo sensu accepti, Lausanne, 1744. 112

[10] G. Fabiani, E. Galaris, L. Russo, and C. Siettos. Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(4), 2023. 111

[11] S. J. Ferraro, D. M. de Diego, and R. T. S. M. de Almagro. Parallel iterative methods for variational integration applied to navigation problems. *IFAC-PapersOnLine*, 54(19):321–326, 2021. 115

[12] Y. Gu and M. K. Ng. Deep neural networks for solving large linear systems arising from high-dimensional problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023. 2, 111, 112

[13] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Verlag, (1993), Second revised edition edition. 111

[14] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*. Springer-Verlag, (1996), Second revised edition edition. 111

[15] C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891, 2019. 17, 18, 117

[16] J.C.S. Kadupitiya, G. C. Fox, and V. Jadhao. Solving Newton's equations of motion with large timesteps using recurrent neural networks based operators. *Machine Learning: Science and Technology*, 3(2):025002, 2022. 111, 112

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 118, 119, 167

[18] S. Kollmannsberger, D. D'Angella, M. Jokeit, and L. Herrmann. *Deep learning in computational mechanics*. Springer, (2021). 17, 18, 19, 111, 112, 117

[19] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998. 112, 132

[20] Y. Li, Z. Zhou, and S. Ying. Delisa: Deep learning based iteration scheme approximation for solving pdes. *Journal of Computational Physics*, 451:110884, 2022. 111, 112

[21] Y. Liu, J. N. Kutz, and S. L. Brunton. Hierarchical deep learning of multiscale differential equation time-steppers, arxiv. *arXiv:2008.09768*, 2020. 111, 112

[22] A. E. H. Love. *A treatise on the mathematical theory of elasticity*. Cambridge University Press, Cambridge, 1863 - 1940. 112

[23] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation

theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021. 111, 131

[24] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021. 111

[25] D. Manfredo, V. Dörlich, J. Linn, and M. Arnold. Data based constitutive modelling of rate independent inelastic effects in composite cables using preisach hysteresis operators. *Multibody System Dynamics*, pages 1–16, 2023. 112

[26] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta numerica*, 10:357–514, 2001. 111

[27] S. Matsutani. Euler's elastica and beyond. *Journal of Geometry and Symmetry in Physics*, 17:45–86, 2010. 112, 113, 116

[28] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas. Hamiltonian neural networks for solving equations of motion. *Physical Review E*, 105(6):065305, 2022. 111

[29] D. Mortari, H. Johnston, and L. Smith. High accuracy least-squares solutions of nonlinear differential equations. *Journal of computational and applied mathematics*, 352:293–307, 2019. 111

[30] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, (1999). 111

[31] K. Ntarladima, M. Pieber, and J. Gerstmayr. A model for contact and friction between beams under large deformation and sheaves. *Nonlinear Dynamics*, pages 1–18, 2023. 112

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 119, 167

[33] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, (2006). 111, 125

[34] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019. 111, 112, 132

[35] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger. On the role of fixed points of dynamical systems in training physics-informed neural networks. *Transactions on Machine Learning Research*, 2022. 112

[36] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, (2003). 111

[37] M. A. Saadat and D. Durville. A mixed stress-strain driven computational homogenization of spiral strands. *Computers & Structures*, 279:106981, 2023. 112

[38] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, and T. Rabczuk. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020. 111, 132

[39] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, and D. Mortari. Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations. *Neurocomputing*, 457:334–356, 2021. 111, 112

[40] D. A. Singer. Lectures on elastic curves and rods. In *AIP Conference Proceedings*, volume 1002, pages 3–32, American Institute of Physics, 2008. 112, 115

[41] M. Stavole, R. T. S. M. de Almagro, M. Lohk, and S. Leyendecker. Homogenization of the constitutive properties of composite beam cross-sections. In *ECCOMAS Congress 2022-8th European Congress on Computational Methods in Applied Sciences and Engineering*, 2022. 112, 132

[42] S. P. Timoshenko and J. M. Gere. *Theory of elastic stability*. McGraw-Hill Book Company, (1961). 116

[43] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 117

[44] L. Vu-Quoc and A. Humer. Deep learning applied to computational mechanics: A comprehensive review, state of the art, and the classics. *Computer Modeling in Engineering & Sciences*, 137(2):1069–1343, 2023. 111

[45] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. 133

[46] E. Weinan and Y. Bing. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018. 111, 132

[47] G. Yagawa and A. Oishi. *Computational Mechanics with Deep Learning: An Introduction*. Springer Nature, (2022). 111

Supervised Time Series Classification for Anomaly Detection in Subsea Engineering

Ergys Çokaj, Halvor Snersrud Gustad, Andrea Leone, Per Thomas Moe and Lasse Moldestad

To appear on the

Journal of Computational Dynamics

Supervised Time Series Classification for Anomaly Detection in Subsea Engineering

Abstract. Time series classification is of significant importance in monitoring structural systems. In this work, we investigate the use of supervised machine learning classification algorithms on simulated data based on a physical system with two states: Intact and Broken. We provide a comprehensive discussion of the preprocessing of temporal data, using measures of statistical dispersion and dimension reduction techniques. We present an intuitive baseline method and discuss its efficiency. We conclude with a comparison of the various methods based on different performance metrics, showing the advantage of using machine learning techniques as a tool in decision making.

6.1 Introduction

In the offshore petroleum industry, drilling, completion and workover of subsea wells is usually performed by semi-submersible drilling rigs. A string of pipe sections extends from the rig to the subsea well and provides a conduit for fluid and tools. To prevent uncontrolled release of oil and gas to the environment this riser system includes a blowout preventer (BOP) directly on the top of the well. The BOP is a heavy steel structure with valves and allows for safe disconnect from the well if needed. A sketch of a BOP stack on a well can be seen in Figure 1 in Section 2.

During operations wave forces acting on the rig, riser and BOP system induce cyclic loading in the uppermost part of the well (the wellhead). This will in turn cause fatigue damage and increase the risk of cracks to develop and grow in critical sections of the wellhead. A total or even partial loss of structural integrity and pressure control due to cracking of the wellhead must be prevented. For this reason great emphasis is placed on predicting and detecting changes in structural response.

During an operation sensor systems may continuously monitor riser and BOP accelerations and the resulting bending moments applied to the wellhead. A systematic change in the relationship between these responses may be an indication of structural failure of the wellhead system. The change may, however, not be easily detectable for a human operator. This paper compares time series classification (TSC) methods for detecting changes in structural response. Several machine learning (ML) algorithms are trained on a synthetic, labelled, data set. Classification is performed either on the raw time series or by first making use of measures of variability of the data, like standard deviation (STD). Being able to classify a labelled data set with time series would serve as a proof of concept for training anomaly detection algorithms to detect cases where a crack occurs.

Our point of departure is a method relying on STD analysis of the data, which we will refer to as the baseline method. In this paper, we investigate and compare a range of alternative statistical approaches and ML techniques for binary classification of time series. We use synthetic, but physically realistic data simulated by a state of the art commercial code and perform our analysis in a supervised learning setting.

The structure of this paper is as follows. In Section 6.2 we summarize the main characteristics of the data set and perform some preliminary analysis, which lays the basis for the following sections. We also introduce a formal definition of the supervised learning classification problem for the given time series data set. We conclude the section with a concise overview of Principal Component Analysis (PCA), one of the most popular dimension reduction techniques, whose theory goes back to Pearson [28] and Hotelling [14]. We use [15] as our main reference.

Sections 6.3-6.7 illustrate five methods to perform the classification task addressed in this work. For each method, we provide a brief description and report on the experimental results.

The baseline method is presented in Section 6.3. This is mainly based on measures of variation of the values in the data set and on regression techniques.

In Section 6.4, logistic regression (LogR) is used on the transformed data from Section 6.2, combined with PCA. LogR was first introduced by Berkson [5] in 1944 and applied to bioassay. Through the years it has been widely used in areas such as biology, medicine, psychology, finance and economics. It has become one of the most used classification algorithms, thanks to its simplicity, efficiency and interpretability, see e.g. [12, 16, 23].

Section 6.5 covers Decision Trees (DTs), a popular supervised classification and regression technique introduced in the 1960s by Morgan and Sonquist in [25]. New concepts, reviews of decision trees and their applications in different fields such as medicine, finance, environmental sciences, are presented in [29, 35, 37].

Section 6.6 illustrates how to use a Support Vector Machine (SVM) [6], an ML algorithm for binary classification of data that continues to be widely popular due to its high performance and robustness to noise. Since the introduction of SVM in 1992 at AT&T Bell Laboratories, it has been applied in fields such as medicine, biology, finance and technology [8].

The last method considered in this paper, investigated in Section 6.7, belongs to the class of deep learning algorithms and uses a Convolutional Neural Network (CNN). Although CNNs were specifically introduced to work with image data [19], thus with input in the form of matrices (tabular data sets), they reached state of the art results also in other fields. In particular, they proved to be effective at capturing patterns in time series, making them among the most

successful deep learning architectures for time series processing [4, 11, 20].

In Section 6.8 we compare the methods based on different accuracy metrics and finally we provide conclusions and discuss research directions in Section 6.9.

Nomenclature	
accx , accy	x and y component of the acceleration
ASM	Attribute Selection Measure
bmx, bmy	x and y component of the bending moment
BOP	Blowout Preventer
CNN	Convolutional Neural Network
DAS	Data Acquisition System
DT	Decision Tree
DWS	Deep Water Strain
FJ	Flex Joint
LogR	Logistic Regression
ML	Machine Learning
MLP	Multi-layer Perceptron
PCA	Principal Component Analysis
SMU	Subsea Motion Units
STD	Standard Deviation
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSC	Time Series Classification
WLR	Wire Load Relief

Table 6.1: List of abbreviations and notations.

6.2 The data set under consideration

The data set at hand is based on simulated data from the Orcaflex software package [26]. This is done due to lack of measurements in the event of a well cracking. The simulated data is obtained from a three-dimensional finite element dynamic analysis in the time domain of the global riser, BOP and wellhead system. The system is exposed to realistic operational loads from a two-dimensional wave energy spectrum based on hindcast data gathered from representative operations. The two-dimensional sea state comprises 200 linear Airy wave components with different combinations of direction, frequency, and

amplitude. In addition to waves, the system is exposed to a statistical median current profile for the same representative area. This is a unidirectional current with velocity varying with depth.

The riser, BOP and wellhead system is represented with one-dimensional line elements with six degrees of freedom. They are modelled with hydrodynamic, hydrostatic and structural properties aimed at giving realistic dynamic load exposure from the environment. This gives a realistic resulting dynamic load and deflection response.

The vessel used for the simulations is stationary, representing a bottom fixed operation vessel, and serves as a fixed reference for the top of the riser. The riser is in constant positive effective tension, with tension magnitude decreasing with water depth. The wellhead is modelled as a composition of line elements, and non-linear force displacement connections with nonlinear lateral force-displacement soil support in the form of P-Y curves, as is recommended practice, see [2] and references therein.

In order to accurately capture the behaviour of intact and broken conditions, the model used in this study is adjusted to match the full three-dimensional solid finite element models of the broken and intact wellhead systems in soil, exposed to representative static loads. The simulation models for the global system and the wellhead calibration model are based on DNV-RP-E104, edition 2019-09 [2].

Sensors logging at 5 Hz are simulated at likely sensor spots, see Figure 6.1. For each sea state two one-hour data sets are created, each based on a simulation with and without a crack in the well, hereby referred to as *broken* and *intact*. The event where a crack occurs has to the authors' knowledge not been measured, nor is it simulated in the data set. Noise is added to the signal based on the sensor accuracy found in the data sheets relative to the in-operation sensors, with only [10] being publicly available. Two other datasets are created with noise multiplied by 10 and 50, to test the robustness of the different methods. Hereby we refer to the three data sets as *Noise 1, Noise 10,* and *Noise 50*.

All of the data is normalised before applying any ML algorithms. Further details on data preprocessing can be found in Appendix 6.A. Although the data observed in real-life operations may have more complex behaviour, we consider the artificial sensor data to suffice as a proof of concept that could be developed further in a later project with data gathered from the field.

Before moving forward, we provide a formal definition of the supervised learning problem addressed in this work. We denote a univariate time series as $X_{\text{uts}} = [x_1, x_2, \ldots, x_n]$, which is an ordered set of real values x_t indexed by integers $t = 1, 2, \ldots, n$, with x_t the value at the t-th discrete time point. We consider X_{uts} as a column vector in $\mathbb{R}^n$. The simulations in our data set are

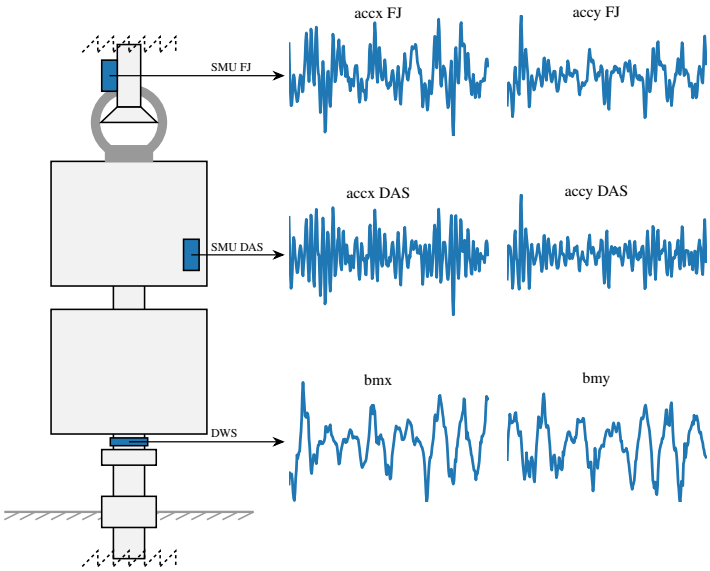


Figure 6.1: Stack with sensors and corresponding data

associated with one-hour long measurements from 3 sensors, sampled at a rate of 5 Hz. Each sensor outputs a signal for the x- and y-direction, hence we have a total of $m = 6$ univariate time series with $n = 18001$ data points. We can collect them in a multivariate time series, which we represent as a matrix

$$X_{\text{mts}} = \left[X^1_{\text{uts}}, X^2_{\text{uts}}, \ldots, X^6_{\text{uts}} \right] \in \mathbb{R}^{n \times m}. \tag{6.2.1}$$

We adopt a supervised learning approach to address the classification problem, as we have access to labelled data. More specifically, the dataset includes N pairs $\mathcal{D} = \left\{ \left(X_i, Y_i \right) \right\}_{i=1}^{N}$, where $X_i \in \mathcal{X}$ are input time series and $Y_i \in \mathcal{Y}$ the corresponding output variables. Here, $\mathcal{X}$ and $\mathcal{Y}$ denote the feature and label domains, respectively. Our aim is to approximate the mapping function

$$F : \mathcal{X} \to \mathcal{Y}, \quad Y_i = F\left(X_i \right), \tag{6.2.2}$$

with sufficient accuracy so that we can make predictions about the output for any unseen input data. To this end, the data set is split 80%−20% into a training- and test-data set. A training procedure is performed on the former set by defining a loss function, that measures the distance between the predictions of the approximation to F and the true labels, and a fitting optimisation algorithm. The accuracy of the approximation is then evaluated on the test set.

In this paper, we deal with a binary classification problem. We map input data into two discrete categories, intact and broken, to which we associate the

labels 0 and 1 respectively, hence $\mathcal{Y} \equiv \{0, 1\}$. Our original data set consists of $N = 103$ multivariate time series, 54 related to the intact case, and 49 to the broken one. Each of them is a collection of 18001 values relative to 6 signals, thus $\mathcal{X} \subset \mathbb{R}^{18001 \times 6}$. The 6 columns of each input data are called channels, and we will also refer to them as the number of input feature maps with a slightly abuse of terminology.

6.2.1 Exploratory data analysis

As we can see in Figure 6.2, it is difficult to separate between an intact or broken well based on a single observation. We do however notice a difference in the spread of the data. This suggests to use a measure of dispersion when classifying.

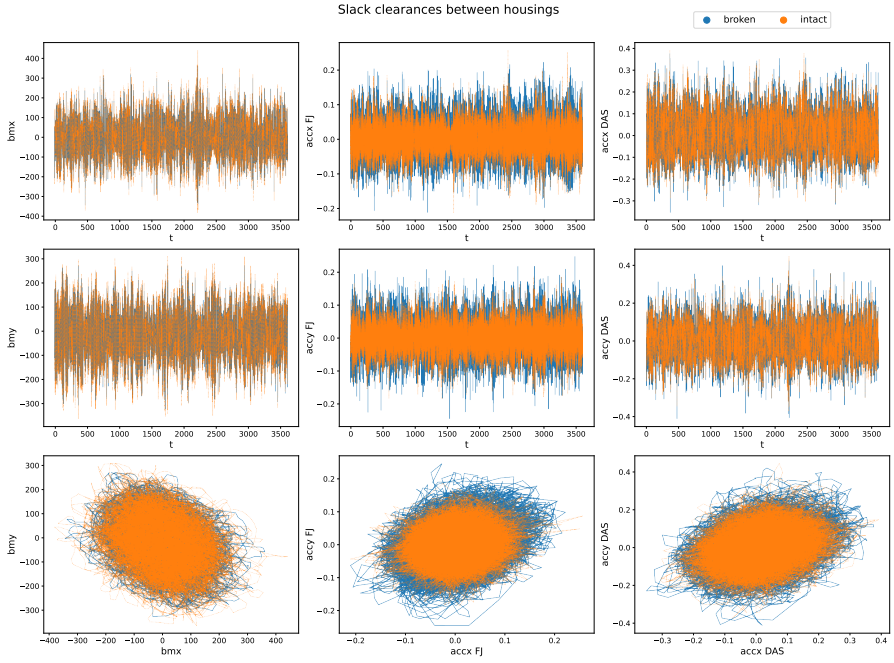


Figure 6.2: Two 1-hour simulations from the dataset comparing a broken and intact well under similar conditions. Plots are given for the x and y component of the different physical measurements. The two top rows give the time series while the bottom row shows phase plots.

Standard deviations transformation

To ensure that a crack in the wellhead is quickly noticed we look into classifying subintervals of the full data set. The simplest dispersion-based classification method consists of taking the standard deviation for each subinterval. More precisely, for each channel m, the standard deviation is calculated over one-minute intervals. Therefore, each one-minute interval with m channels is mapped to a single data point with m dimensions. One-minute intervals allow for updates of the well status at a satisfying frequency while being long enough to give reliable results.

Applying this method to our data set gives us the point clouds found in Figure 6.3. We immediately observe an increased ability to separate the two cases.

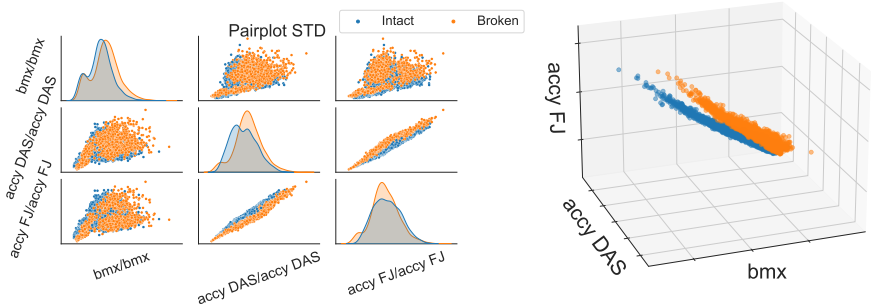


Figure 6.3: Pair plot showing of the scatter and distribution of data after a standard deviation transform (left). Plot visualizing the transformed data in 3 dimensions (right).

Covariance transformation

The standard deviation of the signals can be seen as a meaningful way of separating the data. This suggests that other statistical properties of the signals could be employed. Significant descriptive measures are provided by the covariance and correlation functions [34], therefore we introduce the covariance matrix

$$\Sigma = \begin{bmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \cdots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \cdots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \cdots & \mathrm{Var}(X_n) \end{bmatrix}. \tag{6.2.3}$$

Since we are working with standard deviations, we take the square root of the covariance matrix, given by

$$\Sigma^{\frac{1}{2}} = Q^{\top}\Lambda^{\frac{1}{2}}Q,$$

where Q and Λ store the eigenvectors and eigenvalues of Σ. As standard deviations are implicitly included in the covariance matrix, we highlight that the covariance transform expands the STD transform, thus adding more information.

It is worth noting that the covariance and correlation matrices are closely related since

$$\mathrm{Cor}(X) = \mathrm{diag}(\Sigma)^{-\frac{1}{2}}\ \Sigma\ \mathrm{diag}(\Sigma)^{-\frac{1}{2}}. \tag{6.2.4}$$

For most of the classification methods later presented, the covariance matrix is used, but in Section 6.7 correlation is indirectly utilized.

Given the symmetry of the covariance matrix, only the upper triangular part of the matrix is used in the feature set. If m defines the number of channels, one expects $\frac{1}{2}m(m+1)$ features. For the data set at hand this corresponds to 6 or 21 features, depending on whether one is using one or two physical directions from the sensor output.

In Figure 6.4 we have restricted the data set to one physical direction and plotted a pairwise scatter plot to visualize the transformed data. We observe an increased ability to distinguish between broken and intact compared to the standard deviation method, though the closeness of the point clouds still suggests difficulty in making correct classifications. The main method of transforming the data will mainly be through the use of the covariance matrix.

The attentive reader can also observe that the top left 3×3 block in Figure 6.4 is similar to its corresponding figure with the standard deviation transform. This is to be expected, but underlines that the covariance matrix only adds relevant features.

6.2.2 Principal Component Analysis

PCA is an unsupervised dimension reduction technique that finds patterns or structures in the data and uses them to express the data in a compressed form. This increases the interpretability of multidimensional data while preserving the maximum amount of information and enables its visualization. Preserving the maximum amount of information is equivalent to finding uncorrelated linear combinations of the original data set, called principal components, that successively maximize variance in addition to being uncorrelated with each other. Finding such new variables reduces to solving an eigenvalue-eigenvector problem. More precisely, a data set $\boldsymbol{X}$ is given as input to Algorithm 6.1, provided below. In this work, $\boldsymbol{X}$ will be either the STD- or the COV-transformed

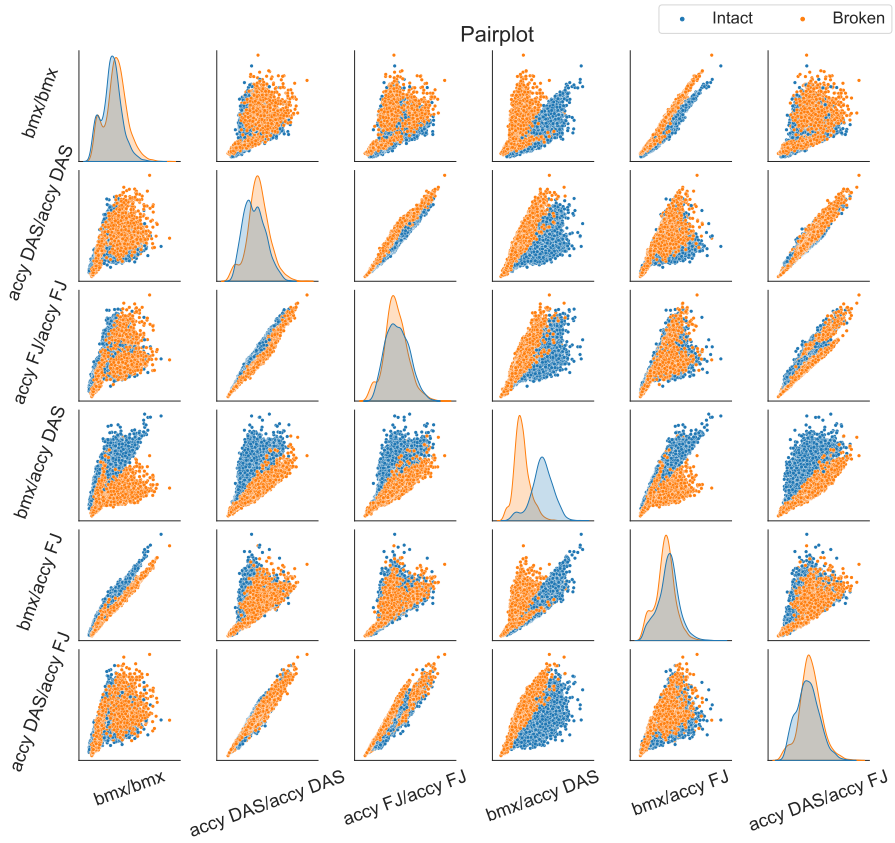


Figure 6.4: Pair plot of the data after using aforementioned covariance transform. For certain combinations the broken and intact cases separate quite well.

data. The algorithm starts by solving an eigenvalue problem for the covariance matrix $\mathbf{\Sigma}$. The $m \times m$ matrix $\boldsymbol{V}$ of eigenvectors diagonalizes the covariance matrix while $\boldsymbol{D}$ is the $m \times m$ diagonal matrix of eigenvalues of $\mathbf{\Sigma}$. The eigenvectors form a basis for the data and the eigenvalues represent the distribution of the information of the source data.

The goal is to choose a small enough subset of d eigenvectors corresponding to the d largest eigenvalues of $\mathbf{\Sigma}$. These will be the new basis vectors onto which we can project the data and still preserve a high quantity of information. This is shown in the final step, where the i-th column of $\boldsymbol{P}$ is the projection of the data points onto the i-th principal component.

Algorithm 6.1 Principal Component Analysis

function $\boldsymbol{P}$ = PCA($\boldsymbol{X}$): ▷ $\boldsymbol{X}$ - input, $\boldsymbol{P}$ - output
  $\boldsymbol{X} \longrightarrow \frac{\boldsymbol{X}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$ ▷ Normalize the data: $\boldsymbol{\mu}$ - mean, $\boldsymbol{\sigma}$ - standard deviation
  $\boldsymbol{\Sigma} = \frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X}$ ▷ Calculate the covariance matrix
  $\boldsymbol{V}^T \boldsymbol{\Sigma} \boldsymbol{V} = \boldsymbol{D}$ ▷ Compute eigenvectors and eigenvalues of $\boldsymbol{\Sigma}$
  $\boldsymbol{W} = \left[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \ldots, \boldsymbol{w}_d\right]$ ▷ Transformation matrix consisting on the first d eigenvectors of $\boldsymbol{V}$ arranged in order of decreasing eigenvalues
  $\boldsymbol{P} = \boldsymbol{X}\boldsymbol{W}$ ▷ Project the data onto the new basis
end function

Figure 6.5 shows the ratio each component explains in the cases when the data is both STD- (left) and COV-transformed (right). In the first case, we see that most of the information is contained in the first 3 components, suggesting one only needs 3 PCs. In the second case, we see that the majority of information is contained in the first 7 components. The accuracy of the method increases along with the number of PCs.

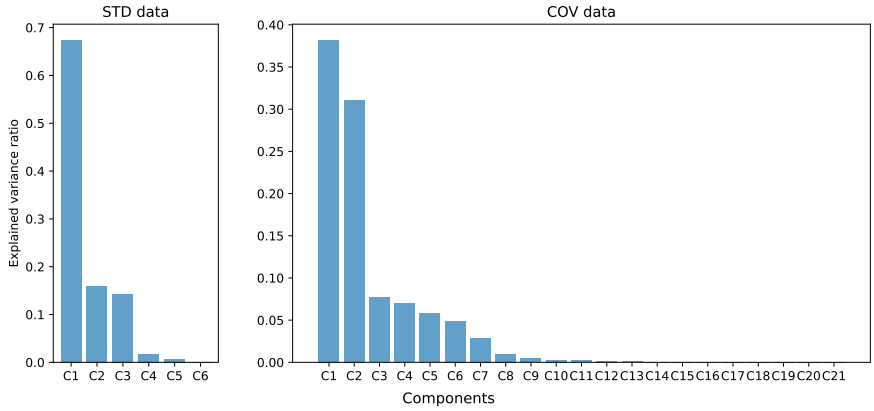


Figure 6.5: Ratio each component explains.

6.3 Baseline method

The baseline method relies on standard deviation and regression, and is currently being used in production. It was designed to enable continuous human inspection and provide an intuitive visual representation of the current behaviour of the wellhead system. This is achieved by drawing regression lines on a monitor.

The method works by sliding a ten-minute window over each of the time series captured by the sensors. The window is split into one-minute intervals for which the standard deviation is calculated. Assume m is the number of sensor channels and let $X \in \mathbb{R}^{m\times 10}$ represent a matrix storing 10 calculated standard deviations for each channel. The method then relies on choosing two rows from X and performing a linear regression. The two rows are typically chosen to be a bending moment and a flex joint acceleration corresponding to the same direction. The regression is given by the following equation

$$\begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix} = \begin{bmatrix}x^\top x & x^\top \mathbb{1}\\ \mathbb{1}^\top x & \mathbb{1}^\top \mathbb{1}\end{bmatrix}^{-1}\begin{bmatrix}x^\top\\ \mathbb{1}^\top\end{bmatrix} y, \tag{6.3.1}$$

where β_0 is the intercept and β_1 is the incline of the regression line, respectively. The ten-minute time window is then moved one time step forward and a new line is drawn. The time step is user defined and is typically set to one minute.

Any significant change between the drawn lines indicates a change in behaviour of the system. Therefore, the occurrence of a crack should be detectable through continuous monitoring of the data. An example of the lines for the cases of a broken/intact well, simulated in a similar environment, can be seen in Figure 6.6. The event where a crack occurs has to the authors' knowledge not been measured, nor is it simulated in the data set.

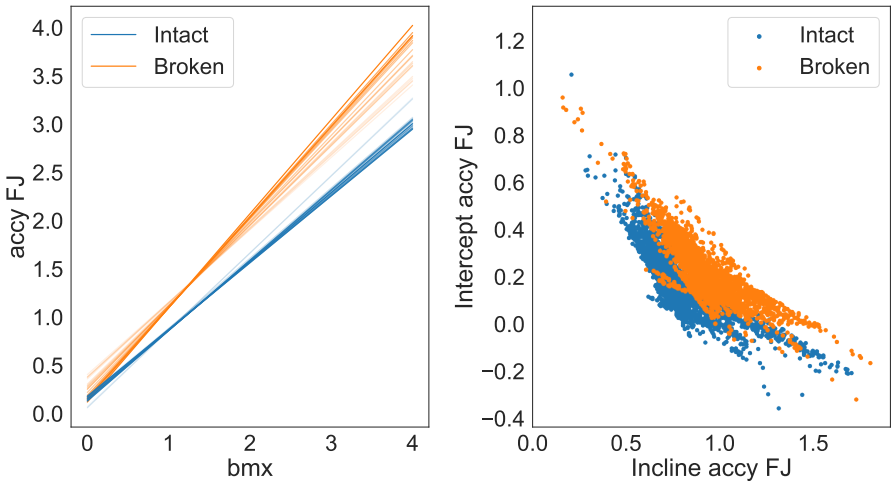


Figure 6.6: Left: Visualization of the lines capturing the relation between the standard deviation of accelerations in the flex-joint and wellhead bending moments using linear regression. The lines are meant to be displayed on a vessel's monitor and gradually fade over time highlighting the most recent behaviour. Right: Distribution of data for the baseline method.

To analyse the method further, we look at the distribution of the intercept and incline of the regression line. The plot to the right in Figure 6.6 illustrates how data points are separated based on whether the well is broken or intact. One may observe a noticeable separation of data, but there is overlap and they lie very close. Similarly to what was observed in Figure 6.4, the closeness of the two distributions suggests difficulty in detecting change in behaviour implying difficulty in classifying the data.

An important feature with this baseline method is the temporal dependence between the lines (left) or points (right) in Figure 6.6. Given the lack of recorded cracking events, we can only speculate on its efficiency. We could however expect a crack to cause the data points to move from their positions in the point cloud representing intact cases to a similar position in the point cloud representing broken. However, given the constraints of our data set, we limit ourselves to examine individual data points whenever a method of dispersion is used.

As a final remark, the linear regression is related to the covariance transform. This becomes clearer when rewriting equation (6.3.1) using the mean, variance and covariance as follows

$$\beta_0 = \mu_y - \frac{\mu_x \operatorname{Cov}(x, y)}{\operatorname{Var}(x)},$$
$$\beta_1 = \frac{\operatorname{Cov}(x, y)}{\operatorname{Var}(x)}.$$

From the equation we read that the baseline method essentially approximates the point clouds from a subplot, depending on the sensor chosen, in Figure 6.4 with a linear regression. The method does however suffer from high uncertainty due to the small set of samples in each prediction.

6.4 Logistic Regression

Given the reduced feature matrix $\boldsymbol{P}$ from Algorithm 6.1 in Section 6.2.2, binary LogR uses a regression technique to solve the two-class classification problem with the class variable $\mathit{Target} = \{\mathit{Broken,\ Intact}\}$ by modelling the class probability $P = \Pr(\mathit{Target} = \mathit{Intact} \mid \boldsymbol{P})$ as

$$\log \frac{P}{1-P} = \beta_0 + \beta^\top \boldsymbol{P}, \tag{6.4.1}$$

with an intercept β_0 and a parameter vector β. The class probability is defined as

$$P = \frac{\exp\left(\beta_0 + \beta^\top \boldsymbol{P}\right)}{1 + \exp\left(\beta_0 + \beta^\top \boldsymbol{P}\right)}. \tag{6.4.2}$$

Fitting a logistic regression model means estimating the intercept β_0 and the parameter vector β. In our experiments, this is done via the `LogisticRegression` from `sklearn.linear_model` with all parameters set to their defaults.

6.4.1 Experiments

In this subsection, we show experiments performed by applying LogR to the reduced feature data set, the output of Algorithm 6.1. We utilize the existing implementation of PCA outlined in Algorithm 1, available through the function `PCA` from `sklearn.decomposition`. We fit `LogisticRegression` to the training set and use the `predict` function to predict the test set result. The LogR-PCA approach is applied to both the STD- and the COV-transformed data from the data set *Noise 1*, *Noise 10*, and *Noise 50*, respectively. For the STD-transformed data, we test the accuracy of the method with the number of PCs going from 1 to 6. In the case of the COV-transformed data, we test for PCs from 1 to 7, since we see from Figure 6.5 that those contain the majority of information. The accuracy of the method in such scenarios, measured with `accuracy.score` of `sklearn.metrics` as the ratio of correctly predicted samples to the total number of samples, is reported in Table 6.2. We see that for the same number of PCs, a higher level of noise leads to a lower accuracy. Hence, to achieve high accuracy even with noisy data, it is necessary to increase the number of PCs. In Figures 6.7 and 6.8, the classification of the time series in the training and test sets is shown for both the STD- and the COV-transformed *Noise 1* data.

Data set and data transformation		Accuracy (%)						
		Number of PCs						
		1	2	3	4	5	6	7
Noise 1	STD	55.99	54.53	69.26	69.17	98.46	98.62	-
	COV	55.24	55.56	65.88	**99.69**	99.84	100	100
Noise 10	STD	55.66	54.53	69.17	69.17	98.14	98.14	-
	COV	55.56	55.87	64.16	**99.53**	99.84	99.84	99.84
Noise 50	STD	54.29	54.21	68.77	69.01	89.97	91.26	-
	COV	55.56	56.81	54.93	**79.34**	91.06	95.62	96.09

Table 6.2: Accuracy of LogR-PCA applied to the STD and COV data from the different data sets with different number of PCs. In bold are marked the scenarios that will be reported in Table 6.6 for comparison purposes.

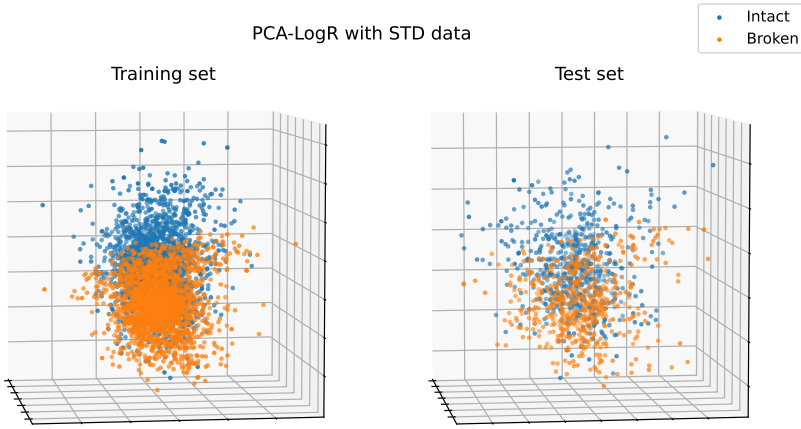


Figure 6.7: Classification of the STD data from the *Noise 1* data set with 3 principal components.

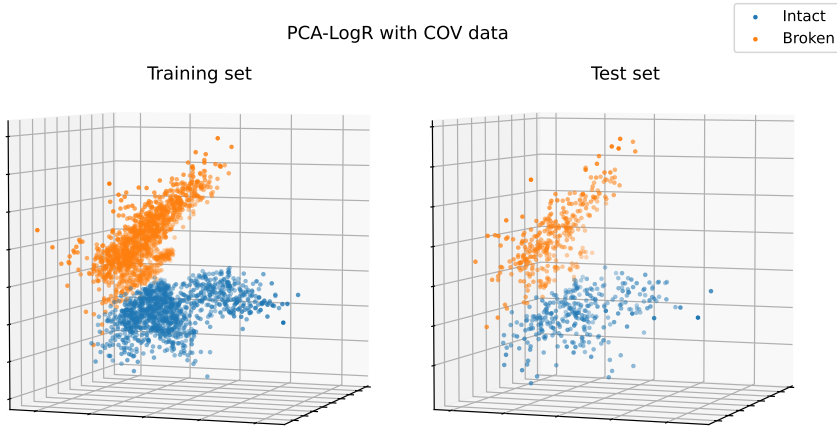


Figure 6.8: Classification of the COV data from the *Noise 1* data set with 4 principal components. The 3D visualization is made with 3 components.

6.5 Decision trees

A decision tree (DT) is a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Given a labelled data set, the model categorizes the data into purer subsets, i.e., subsets consisting of highly homogeneous data, based on a set of if-else conditions. One can think of a DT as a piece-wise constant approximation of the final classification. Figure 6.9 provides some common terminology and illustrates

the idea behind decision trees.

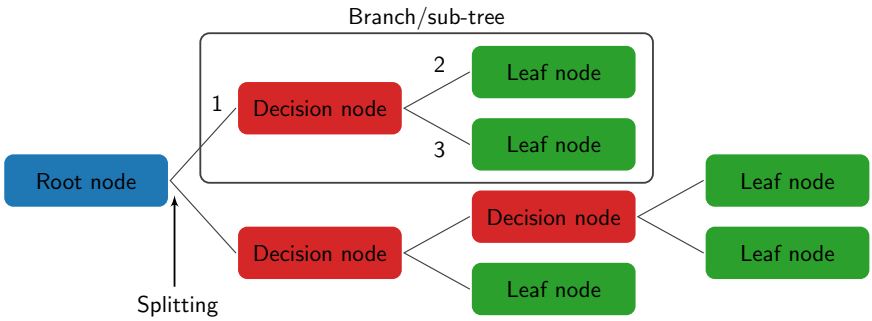


Figure 6.9: Example of a horizontal decision tree with depth 3. Node 1 is the parent node of nodes 2 and 3.

The quality of the splitting, which refers to the purity of the resulting nodes, is measured with Attribute Selection Measure (ASM) techniques. The root node feature is selected based on the results of the ASM, and the procedure is repeated until a node cannot be split into sub-nodes, i.e., until it becomes a leaf node. More specifically, starting from the root node, we evaluate how poorly each feature splits the data into the correct classes, intact or broken. The feature resulting in the lowest impurity is chosen as the best feature for splitting the current node. This is repeated for each subsequent node. There exist two typical ASM techniques for measuring purity, namely *Gini impurity* or *Gini index* and *information entropy* or *information gain*, [24, 32, 36].

The Gini impurity, or the Gini index, (GI) measures the probability of a particular variable being wrongly classified when randomly chosen. In node d, the quantity GI is calculated as

$$GI_d = 1 - \sum_{k=1}^{l} p_{d,k}^2, \tag{6.5.1}$$

where $p_{d,k}$ denotes the probability of an object in node d being classified into the class $k = 1, \ldots, l$. When the parent node d is split, based on a feature f, into m nodes $d_i, i = 1, \ldots, m$, the resulting GI is calculated as the following weighted average:

$$GI_d|_f = \sum_{i=1}^{m} \frac{|d_i|}{|d|} GI_{d_i}, \tag{6.5.2}$$

where $|\cdot|$ denotes the number of data in a node and GI_{d_i} are calculated as in Equation (6.5.1). When this criterion is used for the selection of the root node feature, the feature with the smallest GI is selected. The lower the GI of a node, the closer the node is to being a leaf node. The GI of a pure node is 0.

The information Gain (IG) criterion is based on the entropy (E) measured in each node, which decreases as the purity of the node increases. A pure node has entropy 0. In node d, the quantity E is calculated as:

$$E_d = -\sum_{\substack{k=1 \\ p_{d,k}\neq 0}}^{l} p_{d,k}\log_2(p_{d,k}), \tag{6.5.3}$$

where $p_{d,k}$ is as before. The information Gain (IG) measures the decrease in entropy by computing the difference between entropy before the split and average entropy after the split of the node, based on the chosen feature. Suppose, similarly to above, that the parent node d is split, based on a feature f, into m nodes $d_i, i = 1,\ldots,m$. Then IG of the feature f in node d is calculated as:

$$IG_d|_f = E_d - \sum_{i=1}^{m} \frac{|d_i|}{|d|} E_{d_i}, \tag{6.5.4}$$

where E_{d_i} are calculated as in Equation (6.5.3). The feature yielding the highest IG is chosen as the splitting feature for the node in consideration.

There is no big difference between Gini impurity and entropy when it comes to efficiency, see [31]. The choice varies significantly on the particular circumstances and the data set. One advantage of the GI to the entropy approach is that it does not involve logarithms, which are expensive from a computational point of view. Figure 6.10 shows how the DT algorithm works.

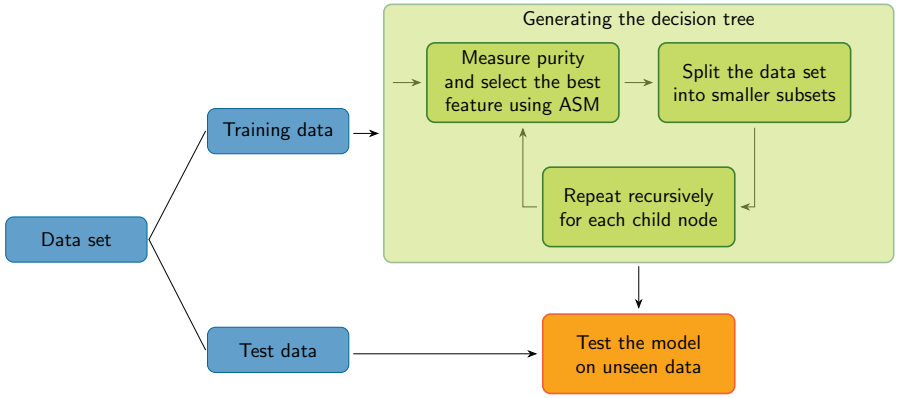


Figure 6.10: Decision tree algorithm illustrated as in [17].

One common difficulty for DTs is overfitting. It can be prevented in two common ways, namely constraining the tree size and pruning the tree, often known as pre-pruning and post-pruning, respectively. Pre-pruning is done by controlling the following parameters: the minimum number of samples required

for a node to split, the minimum number of samples for a leaf node, the maximum number of leaf nodes, the maximum depth of the tree, the maximum number of features to consider while searching for the best split. In post-pruning, nodes and subtrees are replaced with leaves to reduce the complexity of the tree.

6.5.1 Experiments

In the numerical experiments, the trees are generated using the function `tree.DecisionClassifier` from `sklearn` of `Python`, where one can choose between `entropy` or `Gini` splitting `criterion`, and they are displayed using the visualization tool of the `tree` class. `sklearn` uses an optimised version of the CART algorithm [21] which uses `gini` as splitting criterion and considers a binary split for each attribute. When `entropy` is chosen as splitting criterion, the ID3 algorithm [30] is used. Pre-pruning is performed using the function `GridSearchCV` from `sklearn`, which does a thorough search for an estimator over the specific set of parameter values described in the previous section. For the post-pruning, the `cost_complexity_pruning_path` function is used, which is parameterized by the cost complexity parameter `ccp_alpha`. By increasing the value of `ccp_alpha`, the number of pruned nodes increases, and consequently the accuracy decreases, see Figure 6.12. Therefore, one has to make a clever choice of this parameter in order to have significant results. One has to accept a decrease in accuracy in return for a significant reduction in tree complexity.

A series of experiments are run on different scenarios and the results are reported in Table 6.3. The hyperparameter range for the pre-pruning and choice of the α for the post-pruning of the DTs, used to obtain the results reported in Table 6.3, is provided in Appendix 6.B. There is no sign of overfitting of the model in the case of *Noise 1* and *Noise 10* but we notice overfitting in the case of *Noise 50*. We can also see the positive effect of pruning in the reduction of overfitting, in particular when post-pruning. In Figure 6.11, this is shown for the Noise 50, COV-PCA(4) data split with Gini criterion, corresponding to the values in the bottom-right block in Table 6.3. In Figure 6.13, we show the tree generated with `entropy` as splitting criterion applied to the data set consisting of the first four PCs of the COV data. In Figure 6.14, the post-pruned version of the same tree with `ccp_alpha` = 0.01 is shown. The value for `ccp_alpha` is suitably chosen in Figure 6.12. For presentation purposes, the labels are shown only on the root node. The root and decision nodes include the following information: the feature in the data set that best divides the data, the value of the entropy, the number of the samples, their division into the classes and the dominant class, respectively. Leaf nodes are pure and there is no decision to be made.

			Data set											
			Noise 1				Noise 10				Noise 50			
Data transformation	Splitting criterion	DT prun.	Depth	# of nodes	Train acc. (%)	Test acc. (%)	Depth	# of nodes	Train acc. (%)	Test acc. (%)	Depth	# of nodes	Train acc. (%)	Test acc. (%)
STD	Entropy	no	15	454	100	93.69	16	480	100	93.45	19	1018	100	84.39
		pre	13	416	99.47	93.61	13	428	99.37	93.93	12	782	97.17	84.87
		post	12	154	95.57	91.59	12	142	95.31	91.99	10	128	87.15	83.5
	Gini	no	15	530	100	93.45	14	600	100	93.37	19	1078	100	83.25
		pre	13	520	99.84	93.61	13	556	99.55	93.28	10	686	95.47	83.25
		post	10	116	93.75	89.56	10	106	91.42	90.13	9	82	85.17	81.96
COV	Entropy	no	6	48	100	98.28	8	54	100	98.9	11	118	100	94.99
		pre	5	44	99.57	98.28	5	44	98.98	98.75	5	50	95.54	91.86
		post	6	22	98.83	97.97	6	26	98.94	98.59	6	24	95.61	92.8
	Gini	no	7	73	100	98.9	9	80	100	98.59	10	136	100	95.62
		pre	6	60	99.61	99.06	6	62	99.41	98.28	6	76	97.65	95.77
		post	6	26	98.63	98.44	5	24	98.16	98.28	7	32	97.06	95.15
COV-PCA(4)	Entropy	no	**9**	**48**	**100**	**99.22**	8	44	100	98.75	26	792	100	78.72
		pre	5	30	99.61	99.06	7	40	99.92	98.75	**6**	**102**	**83.95**	**82.79**
		post	4	12	99.26	98.75	4	14	98.86	98.9	10	66	83.4	82.0
	Gini	no	9	50	100	98.9	7	58	100	99.37	20	820	100	77.93
		pre	5	34	99.65	98.9	**7**	**58**	**100**	**99.37**	7	206	87.67	80.44
		post	4	12	99.14	98.75	4	12	98.94	99.06	6	24	81.4	79.97

Table 6.3: Performance of DTs tested on different scenarios. In bold are marked the scenarios that will be reported in Table 6.6 for comparison purposes

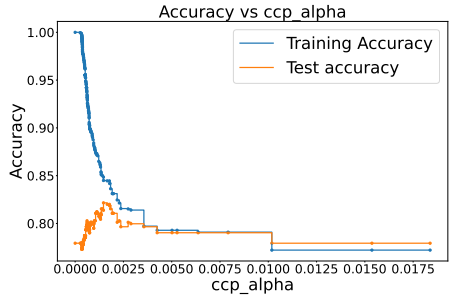


Figure 6.11: The effect of post-pruning in the reduction of overfitting. Scenario: Noise 50, COV-PCA(4), Gini (bottom-right block of Table 6.3.)

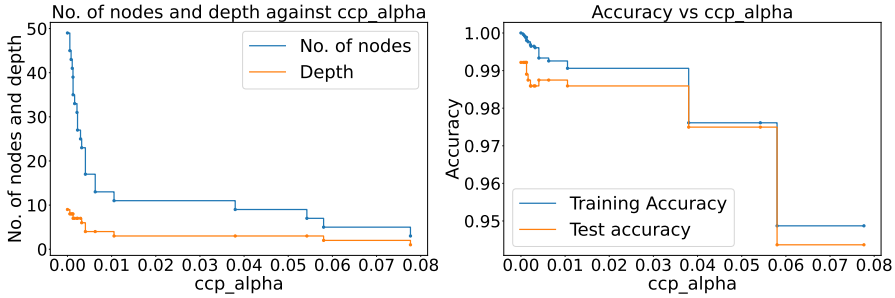


Figure 6.12: The effect of `ccp_alpha` on the structure and the accuracy of the tree. Scenario: Noise 1, COV-PCA(4), Entropy (marked in bold in Table 6.3.)

6.6 Support Vector Machine

Support Vector Machines (SVMs) are ML algorithms that attempt to draw a plane between binary classified data. In the original paper [6], the authors first explain how an optimal hyperplane can be found. This plane can be described as

$$D(x) = \sum_{i=0}^{N} \omega_i \phi_i(x) + b, \tag{6.6.1}$$

where x is the input and ϕ_i is a user-defined basis function. Lastly, ω_i and b are the trainable weights and bias usually found by solving an optimisation problem. The binary classification of the data is based on the sign of the decision function $D(x)$.

The decision function may also be written as

$$D(x) = \sum_{j=0}^{l} y_j \alpha_j K(x_j, x) + b. \tag{6.6.2}$$

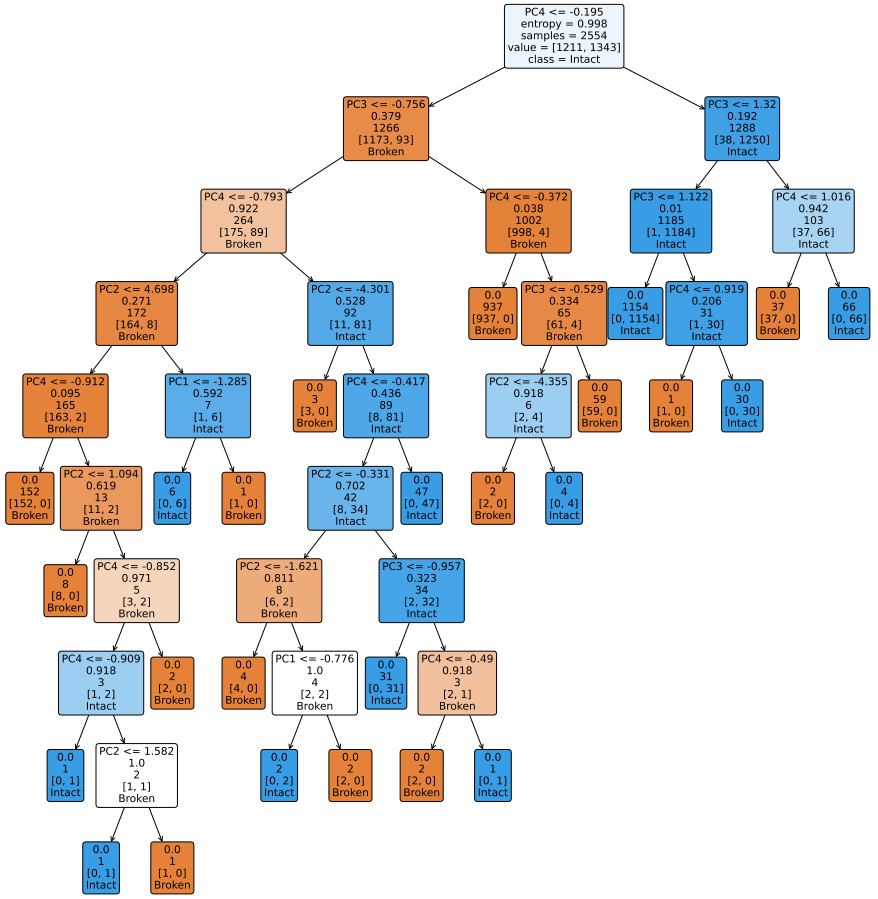


Figure 6.13: DT generated with entropy as splitting criterion on the data set consisting of the first four PCs of the COV data. Blue and orange are used for intact and broken, respectively. A light colour indicates a high entropy, an intense colour a low entropy.

Here, α_i and b are the trainable parameters. The function K is a kernel related to the user functions ϕ_i and x_j are input data. These components are obtained from the dual of the optimisation problem referred to above. In modern software, the kernel is typically defined by the user such that the basis function is never explicitly defined. Commonly used kernels are linear, polynomial and a variety of radial basis functions (RBF).

In [6], the authors demonstrate that training the ML method involves solving a convex quadratic program. The soft margin was later introduced in [9], using l_2-penalization of mislabelled data points, thereby allowing for a feasible

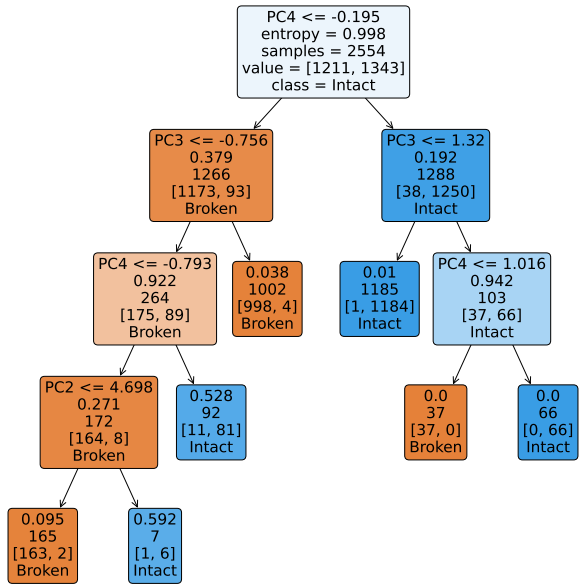


Figure 6.14: The same DT as in Figure 6.13 post pruned with `ccp_alpha` = 0.01.

solution in the case of overlapping classes. Our model is trained by solving the quadratic program that follows,

Primal		Dual	
$\min_{\alpha,\xi,b}$	$\frac{1}{2}\omega^2 + C\xi^\top \mathbb{1}$	$\min_{\alpha}$	$\frac{1}{2}\alpha^\top H\alpha - \alpha^\top \mathbb{1}$
s.t.	$y_i(\omega^\top \phi(x_i) + b) > 1 - \xi_i$	s.t.	$\alpha^\top Y = 0$
	$\xi_i \geq 0$		$0 \leq \alpha \leq C\mathbb{1}$,
	for all i		

and differs slightly from the original method in [6] as it uses l_1-penalization of mislabelled data. Here $Y = \{y_0, \ldots, y_p\}$ are the classifications of the data set, H is an $l \times l$ matrix with elements $H_{ij} = y_i y_j K(x_i, x_j)$. The hyperparameter C allows for a soft margin and ξ_i is the measure of the deviation of point x_i from the margin. Any data point x_i for which the corresponding $\alpha_i > 0$ is considered a support vector. Penalizing the deviations by increasing C increases the number of support vectors, which may lead to overfitting.

6.6.1 Experiments

In this subsection, we evaluate the performance of the SVM through a set of experiments. As in the previous two sections, we apply a dispersion method to transform the data. When the transformation involves the covariance matrix we have also, for comparability between transformations, applied SVM to the top three PCs.

For experiments limited to three dimensions the results are visualised in Figure 6.15. The plots illustrate how a linear plane is able to separate the data points. One can see how the data is relative to the decision border of the linear SVM both for STD transform and COV transform with 3 PCs.

In the experiments, SVMs are trained with either an RBF or a linear kernel. For each choice of kernel, every combination of number of PCs, transformation method and noise level is tested. For each test, the hyperparameter C is optimised using `sklearns GridSearchCV` method. The test accuracy is reported in Table 6.4 along with the number of support vectors needed by the RBF SVMs. For the linear SVM the hyperplane is defined by $n+1$ coefficients, where n is the number of PCs.

Although there is overlap between all the point clouds in Figure 6.15, the PCA based model manages a greater relative distance to the hyperplane, indicating higher robustness. This also becomes apparent by inspecting the number of support vectors for the cases with the same number of PCs, but different transformations, in Table 6.4. The STD based approaches need significantly more support vectors than the COV based, while still performing worse on the test set. SVMs using the COV transform and 7 PCs, essentially spanning the whole data set, only needed a few more support vectors than the ones with 21 PCs. Given that the SVM with RBF kernel relies on a number of support vectors much larger than the number of PCs, it is slower to evaluate than the linear SVM.

6.7 Convolutional Neural Networks

As mentioned in Section 6.2, the supervised learning task consists of estimating the function F in (6.2.2) through a parameterized function F_θ, with θ representing the parameters to be learnt. In this section, we illustrate how neural networks can provide a useful framework to achieve this task.

In the most basic form of fully connected, feedforward neural networks, the input-output mapping F_θ is obtained by a composition of nonlinear functions ϕ:

$$F_\theta(x_0) = \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_l \circ \ldots \phi_1(x_0), \tag{6.7.1}$$

with $x_0 \in \mathbb{R}^{n_0}$ a given input data, L the number of layers in the network, which

Data trans. (#PCs)	Noise 1			Noise 10			Noise 50		
	Linear	RBF		Linear	RBF		Linear	RBF	
	Acc.	Acc.	SV	Acc.	Acc.	SV	Acc.	Acc.	SV
STD(3)*	0.940	0.950	1264	0.866	0.874	1866	0.650	0.668	3591
COV(3)*	0.986	0.986	465	0.974	0.987	568	0.928	0.923	1066
COV(4)*	0.983	**0.990**	418	**0.988**	0.984	441	0.927	**0.940**	980
COV(6)*	0.994	0.999	364	0.983	0.994	444	0.933	0.942	954
STD(6)	0.978	0.983	969	0.926	0.942	1345	0.682	0.726	3239
COV(6)	0.988	0.993	621	0.982	0.994	616	0.946	0.958	992
COV(7)	0.993	0.998	484	0.993	0.996	481	0.953	0.970	853
COV(21)	0.999	1.000	462	0.996	0.998	519	0.947	0.972	923

Table 6.4: Accuracy for linear SVM and RBF SVM applied to the noisy test sets. The number of support vectors for the SVM with the RBF kernel is given in the SV columns. An asterisk (*) indicates that only one physical direction was used from the sensors. In bold are marked the scenarios that will be reported in Table 6.6 for comparison purposes.

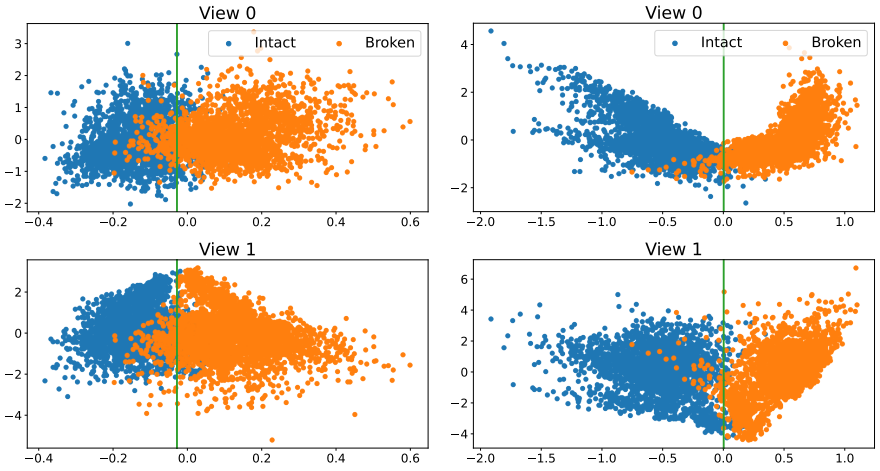


Figure 6.15: Figure showing linear SVMs performance on dataset with STD transform (left column) or COV transform and 3 PCs (right column). Both are created from a subset of the data set containing only one physical direction.

determines its depth, and $\phi_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$, $\phi_l(x_{l-1}) := \sigma\left(W^l x_{l-1} + b^l\right)$ for $l = 1, \ldots, L$. We also refer to these networks as multilayer perceptrons (MLPs). Weight matrices $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ and bias vectors $b^l \in \mathbb{R}^{n_l}$ contain trainable parameters. The nonlinear activation function $\sigma : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$, acting componentwise, typically belongs to C^0 and is monotonically non-decreasing. Examples of such functions are the sigmoid function and the rectified linear unit (ReLU). The training procedure consists of minimising a differentiable loss function, that quantifies the discrepancy between the predictions of the network and the labels, over the network parameters. Usually, a stochastic gradient descent algorithm is used.

Convolutional Neural Networks (CNNs) use particular affine mappings in the feedforward propagation of the input data. In the following, we consider one-dimensional CNNs, where each layer applies a one-dimensional linear kernel K over sections S of the input data, to detect relevant features. Assuming that both the filter K and the receptive field S are defined on the integer i, with $S \in \mathbb{R}^s$ and K having finite support in the set $\{1-s, 2-s, \ldots, s-2, s-1\}$, this operation corresponds to a discrete convolution

$$(S * K)(i) = \sum_{j=1}^{s} S(j)K(i-j).$$

The parameters to be determined during the training are the entries of the linear filters. This results in a significant reduction in parameters, in contrast to dense fully connected neural networks. It should be noted that, reflecting the filter, the convolution operation can be interchanged with correlation. Therefore, since the filter is learnable, its application can also be described in terms of correlation. Input data can include multiple channels, which may vary across different layers. In such cases, the filters are represented by tensors and the convolution operation becomes multidimensional. This allows for the learning of unique features for each channel and the generation diverse feature maps. Each convolutional layer is followed by a pooling layer which uses pooling filters to reduce the dimensionality of the feature maps. The most commonly used pooling techniques are max pooling and average pooling, which, respectively, propagate the maximum and average values from sections of the feature maps [13].

As a result, we can model the forward propagation of the input data in a CNN as a composition of mappings ϕ_{cn} given by

$$\phi_{cn} : \mathbb{R}^{n_{l-1} \times m_{l-1}} \to \mathbb{R}^{n_l \times m_l}, \qquad \phi_{cn}(x_{l-1}) = P(\sigma(C(x_{l-1}))),$$

where n_l and m_l are, respectively, the length and the number of channels of the output tensor of layer l, C is a convolution operator resulting from sliding linear filters across the feature maps from the previous layer and adding a bias, σ is a nonlinear activation function, and P is a pooling operator that coarsens

the grid over which the feature maps are defined [7]. Moving deeper into the network, higher-level features are created. The ones returned from the final pooling layer are usually mapped to a vector and fed to an MLP, which returns a prediction about the class label.

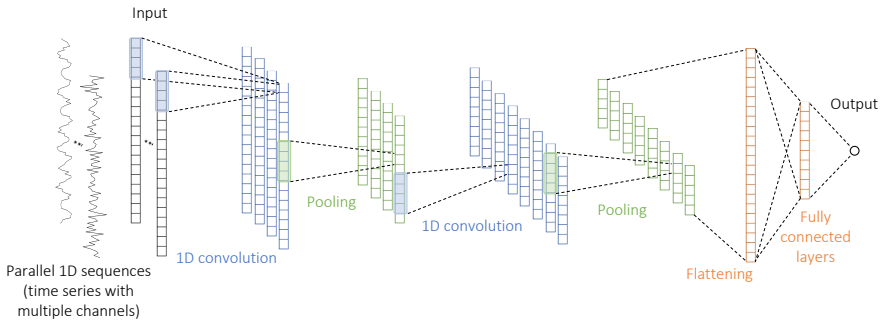


Figure 6.16: A typical one-dimensional CNN architecture.

6.7.1 Experiments

The time series in the original data set were split into one-minute intervals and collected into non-overlapping training and test sets, with the former containing 80% of the resulting series and the latter the remaining 20%. In Figure 6.17, we show the results obtained using a CNN with 3 convolutional layers, each of which doubles the number of channels and is followed by an averaging pooling layer. Finally, an MLP consisting of one hidden layer and an output layer consisting of a sigmoid function is used for prediction. To assign a label to the input data, a threshold is fixed to 0.5, so that when the output is greater than or equal to the threshold, the input time series is classified as broken, or intact otherwise. Details on network architecture can be found in the code snippet listed in Appendix 6.C, written in `PyTorch` [27].

The experiments are run with the number of epochs set to 100. The activation function and certain hyperparameters in the training procedure are varied using the `Optuna` software framework [1]. More specifically, we evaluate different values of batch size, learning rate, and weight decay for the Adam algorithm [18], which is used as optimiser. The specific ranges for each parameter are listed in Table 6.8 in the Appendix. The loss function is defined as the mean squared error (MSE) between the true labels and the predictions of the network. The combinations of hyperparameters yielding the best results on the test set for each level of noise, along with the corresponding mean squared errors on the training and test sets, are presented in Table 6.5.

Selected hyperparameters			
	Noise 1	Noise 10	Noise 50
activation function	LeakyReLU	LeakyReLU	Swish
learning rate η	$2.562 \cdot 10^{-2}$	$2.102 \cdot 10^{-3}$	$1.017 \cdot 10^{-2}$
weight decay	$1.243 \cdot 10^{-5}$	$1.221 \cdot 10^{-5}$	$1.520 \cdot 10^{-7}$
batch size	30	10	30
MSE train	$8.856 \cdot 10^{-6}$	$5.968 \cdot 10^{-5}$	$6.068 \cdot 10^{-4}$
MSE test	$2.815 \cdot 10^{-5}$	$3.054 \cdot 10^{-4}$	$2.427 \cdot 10^{-3}$

Table 6.5: Combination of hyperparameters yielding the best results in each scenario, corresponding to the plots in Figure 6.17, after conducting 100 trials with `Optuna`.

6.8 Comparison of methods

In this section, we compare the tested methods based on performance metrics. We consider precision, recall and F1-score, defined in terms of the entries in the so-called confusion matrix in Figure 6.18 as:

$$\text{Precision} = \frac{TP}{TP+FP}, \quad \text{Recall} = \frac{TP}{TP+FN}, \quad \text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

In Table 6.6, we report the performance of the methods measured with the `Python` functions of `sklearn.metrics`: `classification_report` gives the precision, recall and F1 scores.

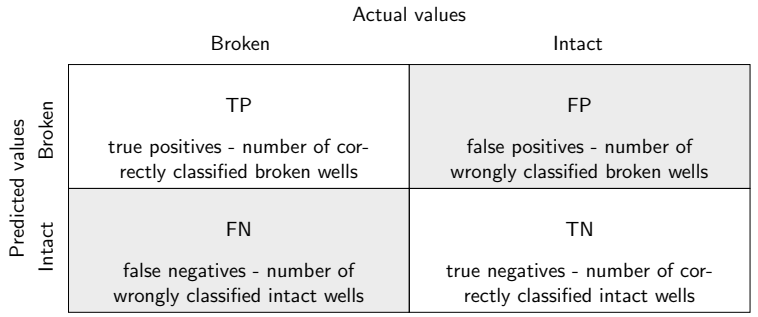


Figure 6.18: Confusion matrix used to evaluate the performance of the classification techniques.

For the methods where we have tested different scenarios, we report here only the best-performing ones, marked in bold in the respective sections. The

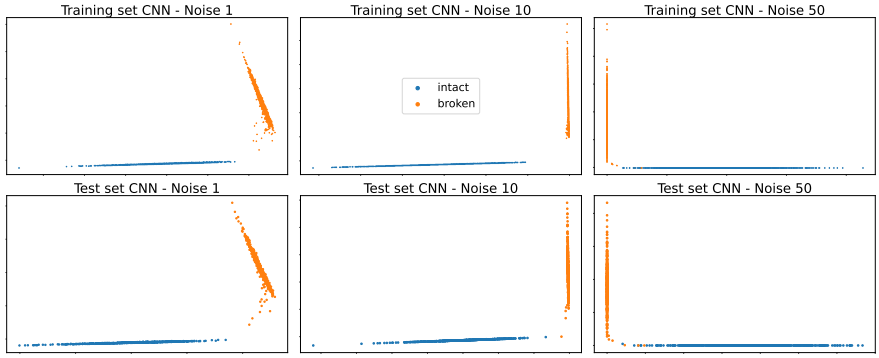


Figure 6.17: The figures illustrate the transformation of the input data by the CNN in both the training and test sets, under the three different noise scenarios. Prior to the output layer, which predicts the class, each individual time series is converted into a two-dimensional vector and can be visually represented as a point on a plane. In the case of Noise 1 and Noise 10, the data points belonging to the two categories form separate clusters.

results indicate that all the classical ML algorithms perform similarly well in terms of accuracy, but are outperformed by the more advanced CNN. For the different methods there are significant differences in the train and test times.

Already with 4 PCs, LogR-PCA shows almost perfect results. The number of parameters needed to make the classifications is only one more than the dimensionality of the data, proving that non-complex algorithms could suffice in classification of the data. The decision trees score second to best using 4 PCs, but needs significantly more parameters than the LogR. As the dimensionality increases so does the number of parameters, making it prone to overfitting. The SVM gets a lower comparative score than the two previously mentioned methods, and needs 940 support vectors. However, as the number of PCs increases, the number of support vectors is reduced, as seen in Section 6.6. This suggests that the SVM would perform better and with higher robustness on a data set with increased dimensionality than e.g. the DTs. Finally, CNN provides the best results in terms of accuracy, and is able to correctly classify all the time series in the Noise 1 and Noise 10 datasets, without requiring pre-processing with PCA and COV-transform. As is common for deep learning algorithms, however, it requires longer offline training time, and a fine tuning of different hyper-parameters.

Data set	Method	Precision	Recall	F1 Score	Train Time (ms)	Test Time (ms)
Noise 1	LogR-PCA	0.997	0.997	0.997	10.195	**0.990**
	DT-PCA	0.997	0.987	0.992	**6.662**	0.998
	SVM-PCA	0.990	0.990	0.990	133.799	51.615
	CNN	**1.000**	**1.000**	**1.000**	~ 3 min	30.535
Noise 10	LogR-PCA	0.997	0.994	0.995	12.408	1.001
	DT-PCA	**1.000**	0.987	0.993	**5.207**	**0.999**
	SVM-PCA	0.988	0.988	0.988	24.639	3.003
	CNN	**1.000**	**1.000**	**1.000**	~ 3 min	27.133
Noise 50	LogR-PCA	0.808	0.750	0.778	11.026	1.016
	DT-PCA	0.830	0.808	0.819	**10.910**	**0.994**
	SVM-PCA	0.940	0.940	0.940	212.493	106.985
	CNN	**0.995**	**1.000**	**0.998**	~ 4 min	49.181

Table 6.6: Performance of the methods. Given the high scoring of the classical ML algorithms on the full data set they are here compared using 4 PCs of the COV-transformed data set.

6.9 Conclusion

We observed in Section 6.2 that measures of statistical dispersion applied to shorter time series are a good preprocessing tool for ML algorithms not specifically designed to handle temporal dependencies. Additionally, we observed how the dimensionality of the COV-transform data set could be significantly reduced using PCA.

We presented in Section 6.3 a baseline method for classifying the time series and discussed its efficiency. Given the method's reliance on human assistance, we were unable to evaluate its performance. However, we found that the method, to a certain extent, would be able to distinguish between broken or intact. Although the method is based on known statistical properties and visualization techniques, making it easy to use for practitioners, it is prone to human error.

In Sections 6.4-6.7, popular ML algorithms were trained on the preprocessed data set. The tests showed that they performed remarkably well. In particular, the good results obtained with a simple and popular method like LogR validates the data transformation in the preprocessing phase. It was observed that the performance of SVM deteriorated faster than LogR and DTs as the dimensionality, i.e., the number of principal components, was reduced. However, the low number of support vectors needed by the SVM with sufficiently high dimensionality makes it a viable choice.

Our findings indicate that classical ML algorithms, even when they are not originally designed to take temporal dependencies into account, can excel in TSC given proper pre-processing. CNNs, on the other hand, suggest that deep learning is a powerful tool to extract discriminative features in time series, without the need of any data manipulation other than normalization. However, a common downside of deep learning algorithms is that the learned features do not have an immediate interpretation. Additionally, when choosing an ML method to be used in production one must carefully weigh the need for computational power versus accuracy.

Given the experimental results, we conclude that ML algorithms are advantageous in order to reduce dependence on human decision making. In future work, it would be of interest to investigate the use of both one-class ML and unsupervised ML algorithms trained on field-measured data, as there are, to the author's knowledge, no documented measurements of a broken well. Such algorithms could be, among others, one-class SVM [33], autoencoders [3], CNN with Long Short Term Memory algorithms [4] or isolation forests [22], which have shown good results for anomaly detection.

Acknowledgments The authors are grateful to Elena Celledoni, Brynjulf Owren and Mathias Hansen for the valuable discussions in various stages of this work. E.Ç. and A.L. would like to thank the group of Structural Analysis Engineering at TechnipFMC Lysaker and Kongsberg for their support and hospitality during their industrial secondment.

Appendix 6.A Data set

In the given maintenance operations, referenced in Section 6.1, the BOP is monitored through the use of Deep Water Strain sensors (DWS) and Subsea Motion Units (SMU). The DWSs give strain values at a cross-section close to the well, which again may be used to calculate loads. The SMUs are used to measure accelerations and rotational velocities above and below the flex joint that connects the riser to the BOP. In certain cases, a load relief system may be applied. One of these is the Wire Load Relief (WLR), which consists of attaching wires to the BOP and securing it to a nearby sturdy structure. Whenever WLR is used, one may also get access to the loads on each wire, but we assume that we do not in this project.

A challenge in this project is that there exist no measurements of a well with a confirmed crack. We model several different cases with an intact and a broken well and analyze the data. The model is set up in the commercial software Orcaflex [26]. The data set we work with is simulated based on a generic well in the North Sea.

When accessing a well, a decision must be made about which tools and configurations to use. This is planned before the start of each operation. Whether one or more configurations will be used varies depending on the operation being carried out. There are, however, specific configurations that, once selected, cannot be changed easily. We set up the data set as follows.

We first consider a realistic combination of permanent configurations based on

- load relief (3 settings),
- drilling or completion (2 settings),
- slack or tight wellhead housing (2 settings).

Other configurations may vary. In our case, we look into

- drillpipe tension (3 settings),
- sea states (18 settings).

Finally, for each combination of the above configurations, two simulations are run with either the well broken or intact. Some settings do not combine and some analyses are not able to converge, hence a total of 987 different analyses are generated, each one hour long. Figure 6.19 gives an overview of the structure of the data set.

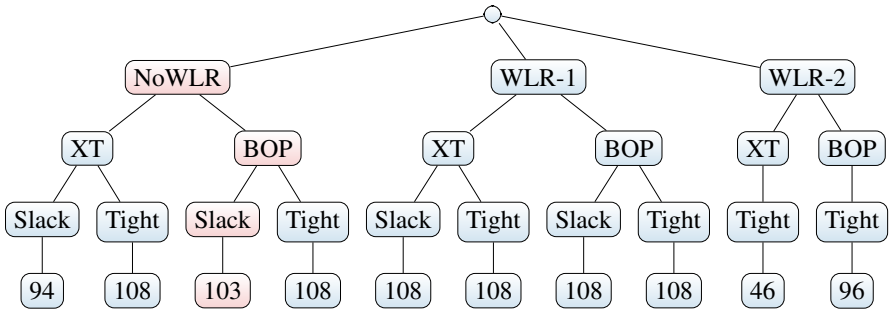


Figure 6.19: Number of analyses for fixed configurations. In red is the combination of configurations that we analyze in this work.

For each analysis, three sensors are simulated at likely sensor positions. Two of these sensors, known as subsea motion units (SMUs), measure acceleration. One sensor measures strains at the wellhead and calculates bending moments, and is known as a deep water strain sensor (DWS). All of these sensors give information about the x- and y-direction and are logging at 5 Hz. A possible setup is shown in Figure 6.1.

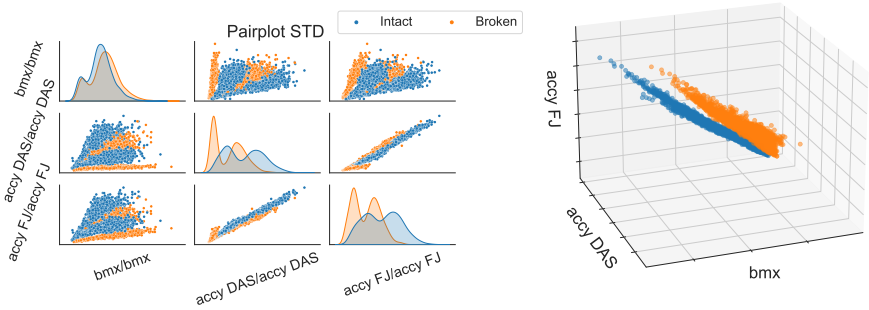


Figure 6.20: To the left a pair plot of the data after using aforementioned standard deviation transform on wells with a tight wellhead housing. For certain combinations the broken and intact cases separate quite well. To the right a 3D plot showing the spread of the data.

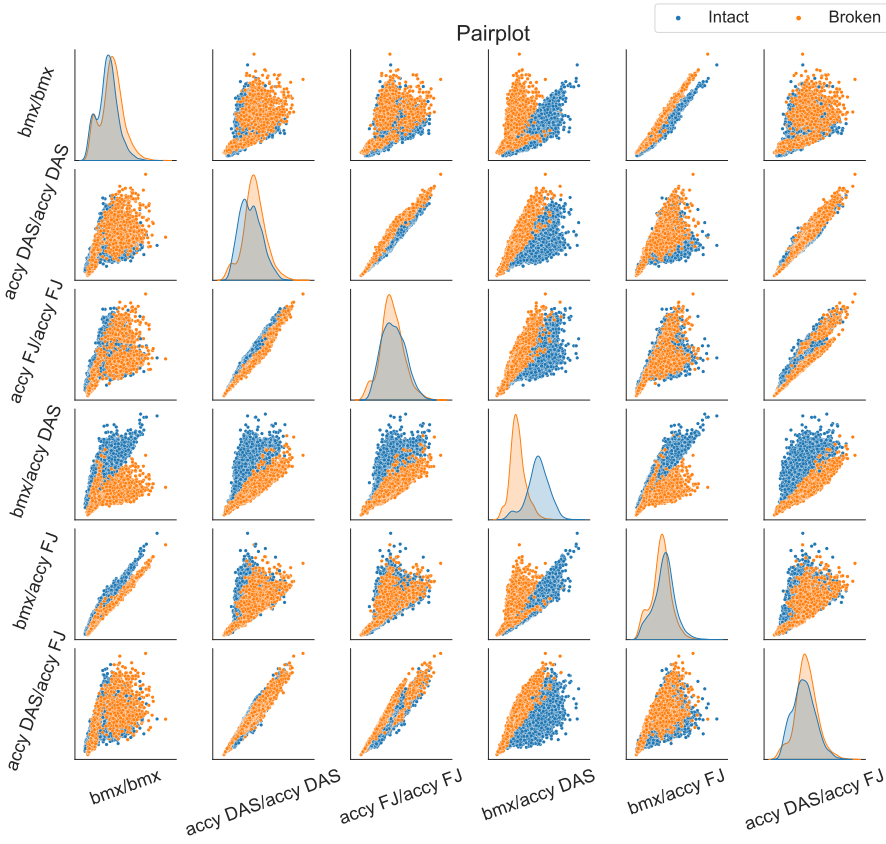


Figure 6.23: Pair plot of the data after using aforementioned covariance transform on wells with a slack wellhead housing. For certain combinations the broken and intact cases separate quite well.

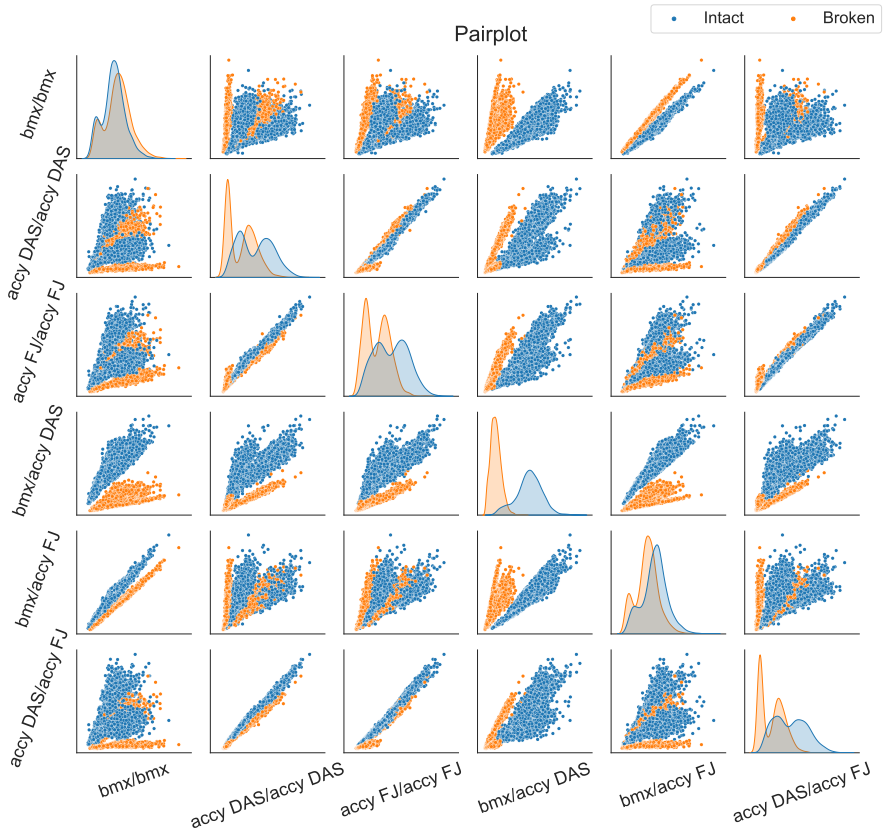


Figure 6.21: Pair plot of the data after using aforementioned covariance transform on wells with a tight wellhead housing. For certain combinations the broken and intact cases separate quite well.

The specific configuration about the wellhead housing (slack/tight) is of particular importance as one might not be sure about this property before accessing the well. If the wellhead housing is slack the BOP is prone to move more around, which is a similar property to a cracked well. In such case we observe an increased difficulty in classifying on slack data. This becomes apparent when we view the data of the slack and tight WH housing in Figure 6.20 to 6.23

Since tight wellhead housing leads to a simpler classification problem than the case with slack, the data set used in the main sections was limited to slack wellhead housing.

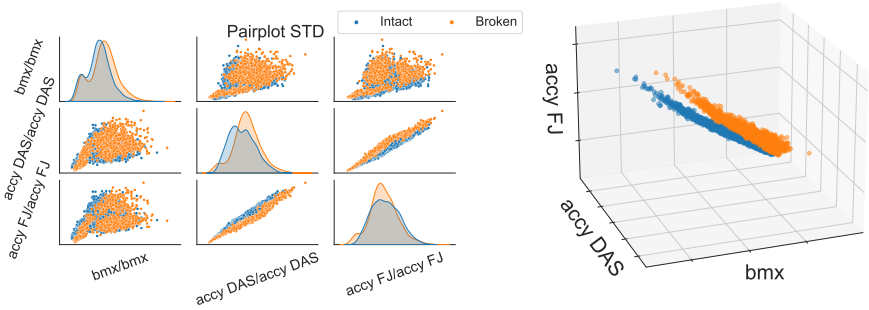


Figure 6.22: To the left a pair plot of the data after using aforementioned standard deviation transform on wells with a slack wellhead housing. For certain combinations the broken and intact cases separate quite well. To the right a 3D plot showing the spread of the data.

6.A.1 Prepocessing the data set

Whether the time series are passed through a transformation described in Section 6.2 or fed directly to the ML algorithm, they need to be pre-processed to improve performance.

To standardize the data set's features to unit scale, i.e., mean equal to 0 and variance equal to 1, we use `StandardScaler` from `sklearn.preprocessing`. We may then apply Algorithm 6.1 to the standardized training and test set, using `PCA` from `sklearn.decomposition`, to reduce the dimensionality.

To train and validate the methods, we divide our data set into a training set and a test set. Typically, these contain 80% and 20% of the original data set, respectively. The machine learning algorithms in this paper makes predictions on the training data and then corrects itself based on the true outputs. Learning stops once the algorithm has achieved an acceptable level of performance on the training set, and the accuracy is measured on the unseen data in the test set.

Appendix 6.B Supplementary material for the reproducibility of the experiments: Decision trees

		Pre-pruning		Post-pruning
Data transformation	Splitting criterion	Hyperparameter	Range	α
STD	Entropy	`max_depth`	$[2,13]\cap\mathbb{N}$	0.003
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	
	Gini	`max_depth`	$[2,13]\cap\mathbb{N}$	0.002
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	
COV	Entropy	`max_depth`	$[2,5]\cap\mathbb{N}$	0.01
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	
	Gini	`max_depth`	$[2,6]\cap\mathbb{N}$	0.003
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	
COV-PCA(4)	Entropy	`max_depth`	$[2,8]\cap\mathbb{N}$	0.01*
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	
	Gini	`max_depth`	$[2,8]\cap\mathbb{N}$	0.003
		`min_samples_split`	$[2,4]\cap\mathbb{N}$	
		`min_samples_leaf`	$[1,2]\cap\mathbb{N}$	

Table 6.7: Hyperparameter ranges for the pre-pruning and choice of the α for the post-pruning of the DTs, used to obtain the results reported in Table 6.3. * except for the *Noise 50* data set where $\alpha = 0.003$.

Appendix 6.C Supplementary material for the reproducibility of the experiments: Convolutional Neural Networks

```
class cnnseries(nn.Module):
    def __init__(self, act_name='lrelu'):
        super(cnnseries, self).__init__()

        torch.manual_seed(1)
        np.random.seed(1)
        random.seed(1)

```

```
        self.conv1 = torch.nn.Conv1d(in_channels = 6,
    out_channels = 12, kernel_size = 30, stride=1, padding=0,
    dilation=1, groups=1, bias=True)
        self.conv2 = torch.nn.Conv1d(in_channels = 12,
    out_channels = 24, kernel_size = 30, stride=1, padding=0,
    dilation=1, groups=1, bias=True, padding_mode='zeros',
    device=None, dtype=None)
        self.conv3 = torch.nn.Conv1d(in_channels = 24,
    out_channels = 48, kernel_size = 30, stride=1, padding=0,
    dilation=1, groups=1, bias=True, padding_mode='zeros',
    device=None, dtype=None)
        self.avgpool = torch.nn.AvgPool1d(kernel_size = 15,
    stride=5, padding=0, ceil_mode=False, count_include_pad=
    True)
        self.fc2 = nn.Linear(2, 1, bias=True, device=None,
    dtype=None)
        self.fc1 = nn.Linear(48, 2, bias=True, device=None,
    dtype=None)
        self.act_dict = {"tanh":lambda x : torch.tanh(x),
                         "sigmoid":lambda x : torch.sigmoid(x),
                         "swish":lambda x : x*torch.sigmoid(x),
                         "relu":lambda x : torch.relu(x),
                         "lrelu":lambda x : F.leaky_relu(x)}
        self.act = self.act_dict[act_name]

    def forward(self, x):
        x = self.act(self.conv1(x))
        x = self.avgpool(x)
        x = self.act(self.conv2(x))
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.act(self.fc1(x))
        x2 = x
        x = torch.sigmoid(self.fc2(x))
        return x, x2
```

Listing 6.1: Architecture of the CNN used in the eperiments of Section 6.7.

Hyperparameter	Range	Distribution
activation function	{Tanh, Swish, Sigmoid, ReLU, LeakyReLU}	discrete uniform
learning rate	$[1\cdot10^{-4}, 1\cdot10^{-1}]$	log uniform
weight decay	$[1\cdot10^{-7}, 5\cdot10^{-4}]$	log uniform
batch size	$\{10, 30, 50, 100\}$	discrete uniform

Table 6.8: Range of values allowed for each hyperparameter in the experiments with CNNs, with the third column describing how the values were explored using `Optuna`.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 167

[2] DNV AS. RECOMMENDED PRACTICE. Technical Report DNV-RP-E104, 01 2019. `https://www.dnv.com/oilgas/download/dnv-rp-e104-wellhead-fatigue-analysis.html`. 146

[3] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders, 2021. 171

[4] M. A. Belay, S. S. Blakseth, A. Rasheed, and P. S. Rossi. Unsupervised Anomaly Detection for IoT-Based Multivariate Time Series: Existing Solutions, Performance Analysis and Future Directions. *Sensors*, 23(5), 2023. 145, 171

[5] J. Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944. 144

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. 144, 161, 162, 163

[7] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. 19, 167

[8] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020. 144

[9] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 9 1995. 162

[10] CTi SENSOR, INC. *TILT – 57A DYNAMIC INCLINOMETER, Three-Axis Accelerometer, Three-Axis Gyroscope*, 2024. `https://ctisensors.com/products/tilt-5x-dynamic-inclinometer/`. 146

[11] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019. 145

[12] F. E. Harrell. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, volume 608. Springer, 2001. 144

[13] Catherine F. Higham and Desmond J. Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019. 166

[14] H. Hotelling. Analysis of a complex of statistical variables into principal components. *J. Educ. Psychol. 24*, pages 417–441, 1933. 144

[15] J. Cadima I. T. Jolliffe. Principal component analysis: a review and recent developments. *Phil. Trans. R. Soc. A 374: 20150202*, 2016. 144

[16] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013. 144

[17] Haesik Kim. *Supervised Learning*, pages 87–182. Springer International Publishing, Cham, 2022. 158

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 118, 119, 167

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 144

[20] D. Lee, S. Malacarne, and E. Aune. Vector quantized time series generation with a bidirectional prior model. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 7665–7693. PMLR, 4 2023. 145

[21] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer, 2000. 159

[22] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. 171

[23] S. Menard. *Logistic regression: From introductory to advanced concepts and applications*. Sage, 2010. 144

[24] Francesco Mola and Roberta Siciliano. A fast splitting procedure for classification trees. *Statistics and Computing*, 7:209–216, 1997. 157

[25] J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, 58(302):415–434, 1963. 144

[26] Orcina Ltd. *Orcaflex*. 145, 171

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 119, 167

[28] K. Pearson. On lines and planes of closest fit to systems of points in space. *Phil. Mag. 2*, pages 559–572, 1901. 144

[29] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman. Decision trees: an overview and their use in medicine. *Journal of medical systems*, 26:445–463, 2002. 144

[30] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986. 159

[31] L. E. Raileanu and K. Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41:77–93, 2004. 158

[32] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005. 157

[33] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 582–588, Cambridge, MA, USA, 1999. MIT Press. 171

[34] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications*. Springer Cham, 2017. 149

[35] Y. Y. Song and L. Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015. 144

[36] Suryakanthi Tangirala. Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications*, 11(2):612–619, 2020. 157

[37] A. Venkatasubramaniam, J. Wolfson, N. Mitchell, T. Barnes, M. Jaka, and S. French. Decision trees in epidemiological research. *Emerging themes in epidemiology*, 14:1–12, 2017. 144

ISBN 978-82-326-8072-6 (printed ver.)
ISBN 978-82-326-8071-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology