Doctoral theses at NTNU, 2024:253

Ergys Çokaj

Structure preserving and machine learning 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

Ergys Çokaj

Structure preserving and machine learning 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

© Ergys Çokaj

ISBN 978-82-326-8098-6 (printed ver.)
ISBN 978-82-326-8097-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2024:253

Printed by NTNU Grafisk senter

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of *philosophiae doctor* (PhD) at the Norwegian University of Science and Technology (NTNU). My PhD project has been part of the European Training Network THREAD. Three years of this project were funded by the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 860124 and one year and two months were funded by the Department of Mathematical Sciences.

I express my deepest gratitude to my supervisor, Brynjulf Owren, for his patience, insightful guidance, and support along the way. Sincere thanks also go to my co-supervisor, Elena Celledoni, with whom I have collaborated very closely in these years. Writing this thesis would have been impossible without their supervision.

During these four years, I have had the chance to spend valuable time collaborating with esteemed groups and individuals across Europe. I have spent one month at the Institute of Applied Dynamics at Friedrich-Alexander-Universität Erlangen-Nürnberg led by Sigrid Leyendecker, one month at the Institute of Mathematics at Martin Luther University Halle-Wittenberg led by Martin Arnold, and three months at the Department of Structural Analysis Engineering at TechnipFMC in Lysaker and Kongsberg led by Per Thomas Moe. I am very thankful to these groups for their hospitality and fruitful collaborations.

I have been fortunate to share this experience with amazing peers both from the THREAD project and from the Department of Mathematics at NTNU. In particular, I would like to express my appreciation to Andrea and Davide for being part of my PhD journey from the first to the last day.

Lastly, I thank my family and my close friends for their continuous encouragement and motivation.

On a personal note, this PhD has been more than an academic pursuit. It has been a journey of personal and professional growth amidst a vibrant and stimulating academic environment. The opportunities for research, industry collaboration, European academic training, and teaching activities have been invaluable. I cherish this unique experience deeply and am excited about how it has prepared me for the future.

Ergys Çokaj
Trondheim, 4 March 2024

Acknowledging AI tools

I acknowledge the use of the following AI tools for the mentioned purposes: (i) `Grammarly` – to correct grammatical mistakes, to rephrase sentences with the purpose of improving clarity and readability, and to avoid using British and American English at the same time, (ii) `ChatGPT` from `OpenAI` – to rephrase sentences with the purpose of improving clarity and readability, and to make figures more aesthetically pleasing.

I declare that I have critically reviewed the feedback generated by Grammarly and ChatGPT and, based on that, I have revised the writing using my own words and expressions. I declare that these AI tools have served only as supportive tools and not as a replacement for my original research and critical thinking.

Contents

Introduction

1.1 Motivation

Highly flexible slender structures like cables, hoses, yarns, or ropes, synonymously called rods or beams, are crucial parts of high-performance engineering systems. They allow the transmission of motion, forces, power and digital information over long distances and through narrow spaces. Their effectiveness comes from an optimal combination of high axial stiffness and high flexibility in the other directions. Their capacity to adapt their geometry to the environment comes with a complex mechanical behaviour which is very difficult to control. The complex mechanical response of such structures in real operational conditions is beyond the capabilities of current modelling tools that are at the core of modern product development cycles. In the European Training Network THREAD, mechanical models and numerical methods have been developed for designing highly flexible slender structures with applications in various fields such as mechanical, aerospace, biomedical, offshore, civil, and textile engineering.

For structures whose one of the dimensions is much larger than the other two, such as cables, ropes and hoses, the Cosserat rod theory suggests decomposing the 3D problem into a 1D global problem describing the centreline position and orientation and a 2D local problem describing the cross-sectional behaviour [1, 24]. Simo and Vu-Quoc did pioneering work in the field of geometrically exact beam formulation in the 1980s [101–103]. These systems are mathematically modelled as differential-algebraic equations (DAEs) evolving on nonlinear spaces with internal and external constraints. The presence of finite rotations along with translational degrees of freedom makes these models more complex. Manifolds and Lie groups have proven to be suitable spaces for the formulation of such systems.

Since the analytical solution of such systems is hardly ever available, one tries to approximate the solution through appropriate numerical integrators. While designing new numerical methods to tackle the resulting highly nonlinear problems, one is interested in obtaining space and time discretisations without destroying the mathematical structure and symmetries of the problem, solving the discrete problem efficiently, guaranteeing the convergence and stability of the numerical solution despite the extreme sensitivity of slender rod models to loading conditions and numerical perturbations, [67]. Developing such geometric time integration algorithms has been the main goal of Work Package 3 of the THREAD project. A full overview of the results achieved by THREAD in this regard can be found at [96] and an open-source code repository is available at [107]. The report [4] describes the developed methods, their theoretical analysis and their numerical behaviour for simple benchmark tests. In the report [67], the contributors provide an excellent literature review of both the available mathematical formulations of these systems and the numerical methods built to integrate them. The literature is broad and we try to mention here important contributions without going into details and with a main focus on integrators constructed on a Lie group setting.

Among the earliest contributions to numerical integration of mechanical systems on manifolds and Lie groups, in particular the rotation group $SO(3)$ and the special Euclidean group $SE(3)$, are the papers from Simo and Vu–Quoc [103] and Lewis and Simo [66]. In [103], a geometrically exact formulation for rods undergoing large motions was developed, and the time stepping was formulated as a version of the Newmark methods [83] applicable to Lie groups. In [66], conservative methods for Hamiltonian systems on Lie groups were constructed. These methods are generalised to the so-called α-methods [47] in a Lie group setting, see [2,3,11].

The foundational work of Moser and Veselov on discrete integrable systems [81] introduced key concepts for the discretisation of mechanical systems, which are often described by their Euler-Lagrange equations or as Hamiltonian systems on manifolds, with or without external forces, [62]. Hamiltonian systems are often formulated on cotangent bundles and, in such cases, symplectic integrators can be derived through the discretisation of a variational problem. This approach is often called discrete mechanics. For Euclidean spaces, this theory was developed in the pioneering work of Marsden and West [74], and it has later been generalised to Lie groups in a series of papers [9,19,31,41–43,61,64,65].

The classical Backward differentiation formula (BDF) methods have played an important role in mechanical engineering, and they were recently generalised to Lie groups [111]. Lie group integrators have been successfully applied for the simulation of mechanical systems, and in bio-mechanics, problems of control and other engineering applications, see, e.g. [22,48,61,89].

The work done in this thesis falls under the contribution to Work Package 3 of THREAD: Geometric numerical methods for rod system dynamics, which developed system-level simulation methods and geometric time integration algorithms able to deal with dynamic interactions between many flexible rods and their environment in large scale models while respecting and preserving the nonlinear mathematical structure of the problem. The manifolds and the Lie groups we have considered in the different papers, such as $SO(3), SE(3), S^2, TS^2, \mathbb{S}^n_{++}$, are all suitable for the formulation of rod systems or mechanical systems similar to them. We provide numerical examples of mechanical systems set on such spaces which are inspired from the modelling of beams and industrial applications.

Outline of the thesis We have divided this thesis into two main parts. We have dedicated the first part to the geometric numerical integration of mechanical systems. This part comprises Papers 1 [17], 2 [16], and 3 [5], corresponding to Chapters 2, 3, and 4, respectively. In the second part, we have used machine learning methods to predict the behaviour of simple examples of beams motivated by medical engineering applications and to detect anomalies in subsea engineering systems based on data. This part consists of Papers 4 [15] and 5 [14], corresponding to Chapters 5 and 6, respectively. In the rest of this chapter, we provide additional background for an easier understanding of the work done in the papers. The chapter ends with a summary of the papers.

1.2 Structure preserving numerical integration

Ordinary Differential Equations (ODEs) serve as a cornerstone in the mathematical modelling of natural phenomena and mechanical systems. Their ubiquity in representing dynamic processes, from the celestial mechanics governing planetary motion to the intricate patterns of biological systems, underscores their significance in both theoretical and applied sciences. However, the complexity inherent in these systems often precludes analytical solutions, compelling the adoption of numerical methods to provide fast and accurate solutions. A rich and vast theory of classical numerical integrators has been developed for this purpose, with Runge-Kutta (RK) and Linear Multi-Step (LMS) methods being the most well-known, [40]. These methods are applied to an Initial Value Problem (IVP) of the form

$$\dot{y}(t) = f\left(y(t)\right) \in \mathbb{R}^n, \quad y(0) = y_0 \in \mathbb{R}^n.$$

While general-purpose numerical techniques offer a means to approximate the solutions of ODEs, they frequently overlook the underlying geometric and structural properties that are intrinsic to many systems. These properties, such as conservation laws and symmetries, play an important role in the long-term behaviour and stability of the numerical solutions. It is here that the field of geometric integration emerges as a pivotal discipline, seeking not only to approximate solutions but to do so in a manner that preserves the qualitative features of the original system. We now present a few results about geometric integration, and we recommend [39] as a good source on the topic for further understanding. We mention here a few examples of structures one could be interested in preserving over time. One of these is volume preservation, a property of divergence-free ODEs. Hamiltonian systems are a subgroup of the divergence-free systems. Numerous volume-preserving numerical methods exist, see e.g. [52, 93, 112]. First integrals, or conserved quantities, are another example. These are functions of the system's state variables that remain constant over time for each trajectory of the system. These quantities can represent physical properties such as energy, momentum, or angular momentum in mechanical systems, and their conservation is often a consequence of the system's underlying symmetries. An introduction to numerical methods that maintain these conserved quantities can be found in [39, Chapter IV]. Discrete gradient methods are a famous group of methods that preserve first integrals along the numerical solution, see e.g. [34, 77]. Further references for integral-preserving methods are [29, 85, 94]. Another example of geometric structure is the preservation of the symplectic 2-form $\omega_0 = dp_i \wedge dq_i$ along the flow of a Hamiltonian ODE. Numerical methods that preserve ω_0 along the numerical solution are called symplectic integrators. There exist several known classes of this type, such as symplectic partitioned Runge–Kutta methods, symplectic splitting methods, variational integrators, and methods based on generating functions, see [39, Chapter VI] and [63, 76, 99]. Also for reversible systems, i.e., systems whose any solution trajectory remains valid even after reversing the direction of time, methods have been developed to maintain this property. A family of reversible numerical methods are the symmetric methods, see [39, Chapter V]. The following example is an illustration of structure preserving numerical integration.

Example 1. *Let us consider the mathematical pendulum with mass m, rod length ℓ, and gravitational acceleration g, all equal to 1. The Hamiltonian of this system is*

$$H(p,q) = \frac{1}{2}p^2 - \cos(q) \tag{1.2.1}$$

and the equations of motion are

$$\dot{p} = -\sin(q), \quad \dot{q} = p. \tag{1.2.2}$$

The quantity (1.2.1) *represents the total energy of the system and is constant along solutions of* (1.2.2)*. This means that the solutions of the system remain on the level curves where they start. In Figure 1.1, we show the phase space of the numerical approximations of the solution obtained with the explicit Euler method and the implicit midpoint rule. One can see how the explicit Euler method yields wrong approximations, drifting away from the level curves of*

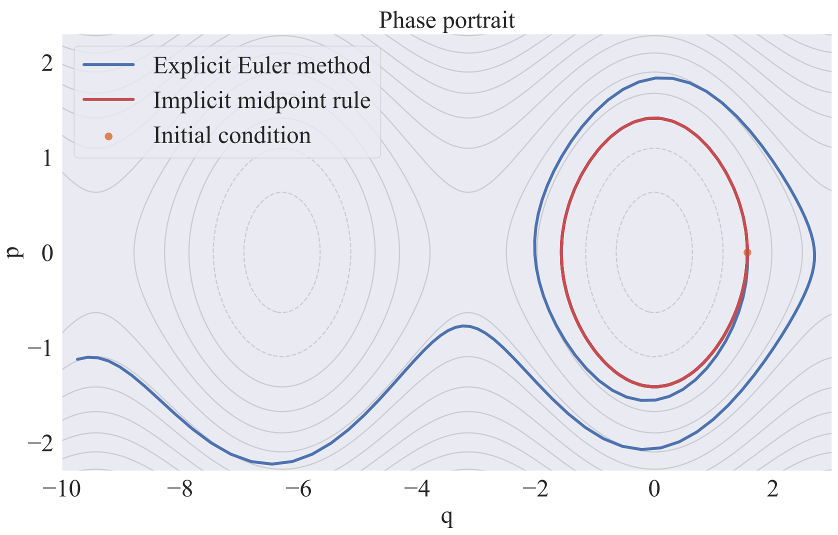

Figure 1.1: Numerical solution of the system (1.2.2) obtained with the explicit Euler method and the implicit midpoint rule. Initial condition $(p_0, q_0) = \left(0, \frac{\pi}{2}\right)$, 120 steps with step size $h = 0.05\pi$, (time $t = 6\pi$). The grey curves represent the exact flow.

$H(p,q)$, while the implicit midpoint rule shows the correct periodic behaviour. Therefore, we can say that the implicit midpoint rule is a geometric integrator for this problem.

1.2.1 Structure preserving integration on manifolds

The dynamics described by the solution of the ODE often takes place on manifolds. One strategy for solving these ODEs is to embed the manifold M into a larger Euclidean space and then apply a classical integrator. This approach has been shown to not preserve the manifold structure and other important geometric properties of the flow of the ODE. This fact has led to a whole new field of study, precisely the structure preserving integration on manifolds, which we have investigated in the first part of this thesis. We illustrate the idea through the following example.

Example 2. *Let us consider a free rigid body with mass centred at the origin, described by Euler's equations*

$$\dot{y} = y \times \mathbb{I}^{-1} y, \tag{1.2.3}$$

where y is the angular momentum vector in the body frame, and $\mathbb{I} = (I_1, I_2, I_3)$ is the inertia tensor. This is an ODE on the manifold defined by the constraint

$$g(y) := \|y\|^2 - R^2 = 0, \tag{1.2.4}$$

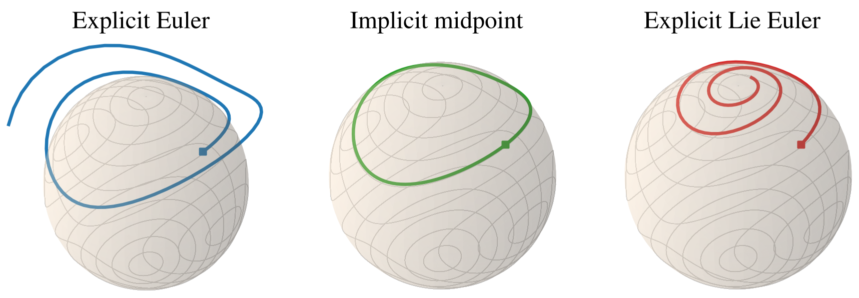

Figure 1.2: Numerical solution of the free rigid body with $I_1 = 2, I_2 = 1.5, I_3 = 1$ approximated with different methods. The square represents the initial condition $y_0 = \left(\cos(0.8), 0, \sin(0.8)\right)^\top$. The explicit Euler and the explicit Lie Euler are integrated with step size $h = 0.1$, while the midpoint rule with $h = 1$. All methods are integrated for 100 steps. The grey solid lines represent the exact flow of the system.

where R is the norm of the initial vector. The flow of (1.2.3) *is energy preserving, i.e., the quantity*

$$H(y) = \frac{1}{2}\left(\frac{y_1^2}{I_1} + \frac{y_2^2}{I_2} + \frac{y_3^2}{I_3}\right), \tag{1.2.5}$$

is conserved over time. Both (1.2.4) *and* (1.2.3) *are quadratic invariants. We can see in Figure 1.2 (left) how the solution obtained with the explicit Euler method not only does not lie on a closed curve, but it even drifts away from the sphere. In fact, most of the classical integrators fail in this setting. The implicit midpoint rule is an exception since it preserves quadratic invariants exactly by construction. Its perfect behaviour can be observed in Figure 1.2 (middle). Lastly, we compute the numerical solution with the simplest Lie group method, i.e., the explicit Lie Euler method. This method works intrinsically on the manifold and ensures that the numerical solution will stay on the manifold. In Figure 1.2 (right), we can see that it indeed perfectly preserves the manifold structure but produces a completely wrong qualitative behaviour. Lie group methods that in addition to the preservation of the manifold structure preserve also the energy are discussed in Chapter 2.*

1.3 Background on Lie groups

This section is dedicated to a more detailed presentation of the concepts and tools of Lie groups and Lie algebras employed in the next chapters. For a more detailed treatment, we refer to [49, 60, 108, 109].

1.3.1 Lie groups and Lie algebras

A *Lie group* G is a differentiable manifold equipped with a product $\cdot : G \times G \to G$ satisfying all group axioms and the smoothness property:

$$\forall p, r \in G,\ \left(p, r\right) \mapsto p \cdot r \ \text{ and } \ p \mapsto p^{-1} \text{ are smooth functions.}$$

A real *matrix Lie group* is a smooth subset $G \subseteq \mathbb{R}^{n \times n}$, closed under matrix products and matrix inversion. In this thesis, we consider only matrix Lie groups and we often refer to them simply as Lie groups.
For $g, h \in G$ the map $L_g : G \to G,\ L_g\left(h\right) = g \cdot h$ is called *left multiplication* and its differential TL_g is the map $TL_g : T_hG \to T_{gh}G$. The notations T_hG and $T_{gh}G$ denote the tangent space of G at h and gh, respectively. Analogously, the map $R_g : G \to G,\ R_g\left(h\right) = h \cdot g$, is called *right multiplication* and its differential RL_g is the map $TR_g : T_hG \to T_{hg}G$. A vector field X is called *left-invariant* if $TL_g \circ X = X \circ L_g$ and *right-invariant* if $TR_g \circ X = X \circ R_g$. A left-invariant vector field is uniquely determined by its value at the identity $e \in G$ of the group as $X|_g = TL_g \circ X|_e$. This identification gives a way to identify the tangent space T_gG at a point $g \in G$ with the tangent space T_eG at the identity $e \in G$.
A *Lie algebra* V is a vector space equipped with a Lie bracket, a bilinear, skew-symmetric mapping $[\cdot,\cdot] : V \times V \to V$ satisfying the Jacobi identity:

$$[u,[v,w]] + [w,[u,v]] + [v,[w,u]] = 0 \quad \forall u, v, w \in V.$$

The Lie algebra $\mathfrak{g}$ of a Lie group G is the linear space of left- (resp. right-) invariant vector fields on G. An alternative definition of the Lie algebra $\mathfrak{g}$ of G is given by taking $\mathfrak{g}$ as the tangent space at the identity element, $\mathfrak{g} := \mathrm{T}_eG$. For a matrix Lie group G, the Lie algebra $\mathfrak{g}$ is the linear subspace $\mathfrak{g} \subseteq \mathbb{R}^{n \times n}$ defined as

$$\mathfrak{g} = \left\{ A \in \mathbb{R}^{n \times n} : A = \left.\frac{\mathrm{d}\rho(s)}{\mathrm{d}s}\right|_{s=0} \right\},$$

where $\rho(s) \in G$ is a smooth curve such that $\rho(0) = I$, with $I \in G$ being the $n \times n$ identity matrix. $\mathfrak{g}$ is closed under matrix additions, scalar multiplication and the matrix commutator $[A, B] = AB - BA$.
Below are a few examples of Lie groups and their Lie algebras (denoted with fraktur fonts):

1. $GL(n) = \left\{A \in \mathbb{R}^{n \times n} : \det(A) \neq 0\right\}$ - the general linear group; $\mathfrak{gl}(n) = \mathbb{R}^{n \times n}$. Every other matrix Lie group is a closed subgroup of $GL(n)$.

2. $SL(n) = \left\{A \in \mathbb{R}^{n \times n} : \det(A) = 1\right\}$ - the special linear group; $\mathfrak{sl}(n) = \left\{A \in \mathbb{R}^{n \times n} : \mathrm{trace}(A) = 0\right\}$.

3. $O(n) = \left\{A \in \mathbb{R}^{n\times n} : A^\top A = I\right\}$ - the orthogonal group;
   $\mathfrak{so}(n) = \left\{A \in \mathbb{R}^{n\times n} : A^\top + A = 0\right\}$ - set of skew-symmetric $n \times n$ matrices.

4. $SO(n) = SL(n) \cap O(n) = \left\{R \in \mathbb{R}^{n\times n} : \det(R) = 1, R^\top R = I\right\}$ - the special orthogonal group, $SO(n) \subset O(n)$; $\mathfrak{so}(n)$.

5. $SE(n) = \left\{\begin{pmatrix} R & v \\ 0 & 1 \end{pmatrix} : R \in SO(n) \text{ and } v \in \mathbb{R}^n\right\}$ - the special Euclidean group.
   $SE(n)$ can be identified as $SE(n) \simeq SO(n) \ltimes \mathbb{R}^n$; $\mathfrak{se}(n) \simeq \mathfrak{so}(n) \ltimes \mathbb{R}^n$, where $\ltimes$ is the semidirect product.

The framework of Lie groups is important because several ODEs evolving on manifolds can be expressed via the concept of a Lie group acting on a manifold. We present such formulation in Section 2.2.1 of Chapter 2. The concepts presented below are of fundamental importance for such formulations.

1.3.2 Lie group action

Let G be a Lie group and M a manifold. A *Lie group (left) action* is a smooth map $\psi : G \times M \to M$ satisfying:

- $\psi(e, m) = m$, $\forall m \in M$, where $e \in G$ is the identity element of G,
- $\psi(g \cdot h, m) = \psi\big(g, \psi(h, m)\big)$, $\forall g, h \in G, m \in M$.

A *right action* is defined as $\psi(g \cdot h, m) = \psi\big(h, \psi(g, m)\big)$. A Lie group action is *global* if $\psi(g, m)$ is defined for all $m \in M$ and $g \in G$ and local if it is defined on an open subset $V \subset G \times M$ such that $\{e\} \times M \subset V$.

Any Lie group G can act on itself by the group multiplication, i.e., with $G = M$, $\psi(g, m) = g \cdot m$. The set $\mathcal{O}_m = \left\{\psi(g, m) \in M :\ g \in G\right\}$ is called the *orbit* of the point $m \in M$ defined by the group action ψ.

Given the action $\psi : G \times M \to M$ of G on M, for each $m \in M$ the *isotropy subgroup at m* is the subgroup $G_m \subseteq G$ consisting of all elements that fix m, i.e., $G_m = \left\{g \in G : \psi(g, m) = m\right\}$. We will see in Chapter 4 that when the isotropy subgroup G_m is nontrivial, i.e., $G_m \neq \{e\}$, there may be more than one element of $\mathfrak{g}$ corresponding to the same vector field in $\mathcal{X}(M)$. This fact can help to obtain better numerical approximations of the solutions of the ODE.

The Lie group action ψ is said to be *free* if for some $m \in M$ and $g \in G$, $\psi(g, m) = m$ implies $g = e$. This is equivalent to saying that $G_m = \{e\}$ $\forall m \in M$. The action is said to be *effective* if $\psi(g_1, m) = \psi(g_2, m)$, for $g_1, g_2 \in G$ and $\forall m \in M$ if and only if $g_1 = g_2$, or in other words, if the only element of G that fixes every element of M is the identity e. ψ is said to be *transitive* if

$\forall x, y \in M$, $\exists g \in G : y = \psi(g, x)$, or equivalently if the orbit of each point in M coincides with the whole of M. In this case, M is called a *homogeneous space*. Any transitive Lie group action corresponds to a homogeneous space and viceversa.

Theorem 1.1 ([88, Theorem 2.17]). *A Lie group G acts globally and transitively on M if and only if $M \simeq G/H$ is isomorphic to the homogeneous space obtained as G/H with $H = G_m$ the isotropy subgroup of any chosen $m \in M$.*

Example 3 (The n-sphere). *The n-sphere $M = S^n = \left\{y \in \mathbb{R}^{n+1} : y^\top y = 1\right\}$ is the homogeneous space related to the left action given by the matrix-vector product $\psi(A, y) = Ay$, $A \in SO(n+1)$. S^n can be obtained as $S^n \simeq SO(n+1)/SO(n)$. The case $n = 2$ is exploited in Chapter 4.*

Example 4 (The spherical pendulum). *In Chapter 2, we have expressed the dynamics of the spherical pendulum in terms of a transitive action by $SE(3)$ on TS^2. We then generalise this analysis to the case of a chain of spherical pendula, which can be seen as a simplified version of a discretised beam. A more realistic formulation of a beam-type system set on the same configuration space is that of the geometrically exact fixed-free elastic rod presented in [62, Section 10.5]*

Two important constructions in this setting are the adjoint and the coadjoint representations of a Lie group G. These are ways of representing the elements of the Lie group as linear transformations of its Lie algebra or its dual, respectively, considered as vector spaces. We discuss these constructions in Example 10 in Chapter 2, where we consider the case $G = SO(3)$.

1.3.3 Exponential map

The *exponential map* $\exp : \mathfrak{g} \to G$ is the most important instrument linking a Lie group and its Lie algebra. For $X \in \mathfrak{g}$, it is defined as $\exp(X) = x(1)$ with $t \mapsto x(t) \in G$ such that $\dot{x}(t) = X|_{x(t)}, x(0) = e$. The exponential map is the flow of right- or left-invariant vector fields. For matrix Lie groups, $\exp : \mathfrak{g} \to G$ is defined as

$$\exp(A) = \sum_{j=0}^{\infty} \frac{A^j}{j!}, \quad A \in G. \tag{1.3.1}$$

For certain matrix Lie groups, a closed form of (1.3.1) is known. We mention here the case of $SO(3)$ and $SE(3)$ which are groups of high importance in applications to mechanics.

For the map $\exp : \mathfrak{so}(3) \to SO(3)$, a closed form is given by

$$\exp(\widehat{\omega}) = I + \frac{\sin\theta}{\theta}\widehat{\omega} + \frac{1-\cos\theta}{\theta^2}\widehat{\omega}^2, \tag{1.3.2}$$

where I is the 3×3 identity matrix, $\theta = \|\omega\|$, and $\widehat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \mathfrak{so}(3)$, with $\widehat{\cdot} : \mathbb{R}^3 \to \mathfrak{so}(3)$. This is known as Rodrigues formula [73].

For the map $\exp : \mathfrak{se}(3) \to SE(3)$ let us first introduce the vector $v = \left(\begin{smallmatrix} t \\ \omega \end{smallmatrix}\right) \in \mathbb{R}^6$ and the matrix $A(v) = \left(\begin{smallmatrix} \widehat{\omega} & t \\ 0 & 0 \end{smallmatrix}\right) \in \mathbb{R}^{4\times 4}$, where t and ω are the translation and the rotation 3-vectors, respectively. Then, the exponential of $A(v)$ is obtained as

$$\exp\left(A(v)\right) = \begin{pmatrix} \exp\left(\widehat{\omega}\right) & Vt \\ 0 & 1 \end{pmatrix}, \tag{1.3.3}$$

$$V = I + \frac{1-\cos\theta}{\theta^2}\widehat{\omega} + \frac{\theta - \sin\theta}{\theta^3}\widehat{\omega}^2, \tag{1.3.4}$$

with I and θ as before and $\exp\left(\widehat{\omega}\right)$ as in (1.3.2).

For the construction of numerical schemes, the derivative of the exponential map $\left(\text{dexp}\right)$ and its inverse $\left(\text{dexp}^{-1}\right)$ are employed. These tools are important because they allow the numerical approximation to lie exactly on the manifold, i.e., they ensure the preservation of the manifold structure. We show in Chapter 2 the general formula to write these functions as a series of nested Lie brackets [45], using the ad-operator $\text{ad}_u : \mathfrak{g} \to \mathfrak{g}$, $v \mapsto [u, v]$. Computing the exact expressions of these functions for the particular Lie algebra in use is also possible. The exact expression for $\left(\text{dexp}^{-1}\right)$ of $\mathfrak{so}(3)$ was computed in [20]. We compute the exact expression for $\left(\text{dexp}^{-1}\right)$ of $\mathfrak{se}(3)$ in Chapter 2 and discuss alternatives to these maps when they become computationally infeasible, as is the case for Lie algebras of larger dimensions.

1.4 Background on Riemannian manifolds

In this section, we give a more detailed presentation of the concepts and tools of Riemannian geometry, mostly adhering to [60].

1.4.1 Riemannian manifolds

A *Riemannian manifold* is a pair $\left(M, g\right)$ where M is a smooth manifold and g, called *Riemannian metric*, is a second order symmetric tensor on TM that assigns to each point $p \in M$ a positive-definite inner product $g_p : T_pM \times T_pM \to \mathbb{R}$. We use interchangeably the notations $g\left(\cdot,\cdot\right)$ and $\langle\cdot,\cdot\rangle$ for the inner product, and $\left(M, g\right)$ and M for the Riemannian manifold in consideration. A parametrised curve $\gamma : [a, b] \to M$ is called an *admissible curve* if there exists a partition $\left(a_0, \ldots, a_k\right)$ of $[a, b]$, $a = a_0 < a_1 < \cdots < a_k = b$, such that $\gamma(t)$ is smooth and $\dot{\gamma}(t) \neq 0$ on each subinterval $\left[a_{i-1}, a_i\right]$ of the partition for $i = 1, \ldots, k$. In this thesis, we consider only such curves. Given a metric g, we can define the length of an admissible curve $\gamma : [a,b] \to M$ as $\ell(\gamma) = \int_a^b \|\dot{\gamma}(t)\| \, dt$, where $\|v\| := \langle v, v\rangle^{\frac{1}{2}}$ is the norm defined by g, or the g-norm. The metric induces a distance function between pairs of points $p, q \in M$, $d(p,q) = \inf_{\gamma_{p\to q}} \ell\left(\gamma_{p\to q}\right)$, where $\gamma_{p\to q}$ is any admissible curve connecting p and q.

Every smooth manifold admits a Riemannian metric. Lie groups are a large class of homogeneous Riemannian manifolds. A Riemannian metric g on a Lie group G is called *left- (resp. right-) invariant* if it is invariant under left (resp. right) multiplication, i.e., turns left (resp. right) multiplication into isometries. It is called *bi-invariant* if it is invariant under both left and right multiplications. Every Lie group admits a left- or right-invariant metric. Additionally, every compact Lie group admits a bi-invariant metric, see [60, pg. 67-72]. To furnish G with a Riemannian metric g one can impose an inner product $\langle \cdot, \cdot \rangle_e$ in $\mathfrak{g} := T_eG$ and then extend it to a left-invariant metric on G by defining for each $x \in G$ and all $u_x, v_x \in T_xG$

$$g\left(u_x, v_x\right) = \left\langle TL_{x^{-1}} u_x, TL_{x^{-1}} v_x \right\rangle_e .$$

Example 5 ($G = SO(n)$)**.** *Let $G = SO(n)$. For $x \in SO(n)$ and all $u_x, v_x \in T_xSO(n)$, a bi-invariant Riemannian metric g in G is $g\left(u_x, v_x\right) = trace\left(u_x^\top v_x\right)$.*

1.4.2 Connections, parallel transport, torsion, and curvature

Connections are a tool to compare values of a vector field at different points, i.e., they serve as a rule for computing directional derivatives of vector fields. This can be seen intuitively as connecting the nearby tangent spaces where these values of the vector field live. Formally, a *connection* ∇ on TM, also called an affine connection, is a map

$$\nabla : \mathcal{X}(M) \times \mathcal{X}(M) \to \mathcal{X}(M), \quad (X, Y) \mapsto \nabla_X Y,$$

that satisfies the following properties:

- linearity over $C^\infty(M)$ in X : for $f_1, f_2 \in C^\infty(M)$ and $X_1, X_2 \in \mathcal{X}(M)$,

$$\nabla_{f_1 X_1 + f_2 X_2} Y = f_1 \nabla_{X_1} Y + f_2 \nabla_{X_2} Y,$$

- linearity over $\mathbb{R}$ in Y : for $a_1, a_2 \in \mathbb{R}$ and $Y_1, Y_2 \in \mathcal{X}(M)$,

$$\nabla_X \left(a_1 Y_1 + a_2 Y_2\right) = a_1 \nabla_X Y_1 + a_2 \nabla_X Y_2,$$

- the Leibniz product rule: for $f \in C^\infty(M)$,

$$\nabla_X (fY) = f \nabla_X Y + (Xf)\, Y.$$

$\nabla_X Y$ is called the *covariant derivative of Y in the direction of X*. Since the covariant derivative $\nabla_X Y$ of a vector field Y at a point p depends only on the value of the vector field X at p, one can define the covariant derivative along a smooth curve $\gamma(t)$ in a manifold, $D_t Y = \nabla_{\dot{\gamma}(t)} Y$. A vector field $Y \in \mathcal{X}(M)$ along a smooth curve γ is said to be *parallel along* γ with respect to ∇ if $D_t Y = 0$ for all t. Every tangent vector at any point on a curve can be uniquely extended to a parallel field along the entire curve.

Theorem 1.2 (Parallel Transport, [60, Theorem 4.31])**.** *Let M be a smooth manifold, and ∇ a connection in TM. Given a smooth curve $\gamma : I \to M, t_0 \in I$, and a vector $v \in T_{\gamma(t_0)}M$, there exists a unique parallel vector field V along γ such that $V(t_0) = v$. V is called the parallel transport of v along γ.*

Parallel transport determines covariant differentiation and the connection, see [60, p. 105-110].

A connection ∇ on TM defines two tensor fields, the *torsion* tensor and the *curvature* tensor. The torsion tensor is defined as

$$\begin{gathered}(X,Y) \mapsto T(X,Y), \quad T : \mathcal{X}(M) \times \mathcal{X}(M) \to \mathcal{X}(M), \\ T(X,Y) = \nabla_X Y - \nabla_Y X - [X,Y].\end{gathered} \tag{1.4.1}$$

The *curvature tensor* is defined as

$$\begin{gathered}Z \mapsto R(X,Y)Z, \quad R : \mathcal{X}(M) \times \mathcal{X}(M) \times \mathcal{X}(M) \to \mathcal{X}(M), \\ R(X,Y)Z = \nabla_X \nabla_Y Z - \nabla_Y \nabla_X Z - \nabla_{[X,Y]} Z,\end{gathered} \tag{1.4.2}$$

and is often called the *Riemann curvature endomorphism*. The curvature tensor R can be interpreted as the measure of the failure of parallel transports around an infinitesimal loop to come back exactly to the original vector.

1.4.3 The Levi-Civita Connection

On every Riemannian manifold there is a natural connection which plays an important role. It is called the *Levi-Civita connection* and it is the only connection ∇ that is *compatible with the metric* g, in the sense that it satisfies the following product rule for all $X, Y, Z \in \mathcal{X}\left((M,g)\right)$:

$$\nabla_X \langle Y, Z \rangle = \langle \nabla_X Y, Z \rangle + \langle Y, \nabla_X Z \rangle, \tag{1.4.3}$$

and *torsion-free*, i.e.,

$$T(X,Y) = 0. \tag{1.4.4}$$

Condition (1.4.4) is equivalent to the requirement of ∇ being *symmetric*, i.e., for all $X, Y \in \mathcal{X}\left((M,g)\right)$

$$\nabla_X Y - \nabla_Y X = [X,Y]. \tag{1.4.5}$$

The existence and the uniqueness of a connection satisfying the properties mentioned above are summarised in the so-called *Fundamental Theorem of Riemannian Geometry*:

Theorem 1.3 ([60, Theorem 5.10]). *Let (M, g) be a Riemannian manifold. There exists a unique connection ∇ on TM that is compatible with g and symmetric. It is called the Levi-Civita connection of g and is uniquely determined by the so-called Koszul's formula:*

$$\begin{aligned}\langle \nabla_X Y, Z\rangle = \frac{1}{2}\Big(& X\langle Y, Z\rangle + Y\langle Z, X\rangle - Z\langle X, Y\rangle \\ & - \big\langle Y, [X, Z]\big\rangle - \big\langle Z, [Y, X]\big\rangle + \big\langle X, [Z, Y]\big\rangle\Big).\end{aligned} \tag{1.4.6}$$

Proof. We only include the proof for the uniqueness here and refer to [60] for the existence. Suppose, that such a connection ∇ exists and let $X, Y, Z \in \mathcal{X}(M)$. Writing the compatibility equation three times with X, Y, Z cyclically permuted, one has

$$X\langle Y, Z\rangle = \langle \nabla_X Y, Z\rangle + \langle Y, \nabla_X Z\rangle, \tag{1.4.7}$$
$$Y\langle Z, X\rangle = \langle \nabla_Y Z, X\rangle + \langle Z, \nabla_Y X\rangle, \tag{1.4.8}$$
$$Z\langle X, Y\rangle = \langle \nabla_Z X, Y\rangle + \langle X, \nabla_Z Y\rangle. \tag{1.4.9}$$

Using the symmetry condition on the last term in each line, we have

$$X\langle Y, Z\rangle = \langle \nabla_X Y, Z\rangle + \langle Y, \nabla_Z X\rangle + \langle Y, [X, Z]\rangle, \tag{1.4.7$'$}$$
$$Y\langle Z, X\rangle = \langle \nabla_Y Z, X\rangle + \langle Z, \nabla_X Y\rangle + \langle Z, [Y, X]\rangle, \tag{1.4.8$'$}$$
$$Z\langle X, Y\rangle = \langle \nabla_Z X, Y\rangle + \langle X, \nabla_Y Z\rangle + \langle X, [Z, Y]\rangle. \tag{1.4.9$'$}$$

Adding (1.4.7′) and (1.4.8′) and subtracting (1.4.9′) we obtain

$$\begin{aligned}& X\langle Y, Z\rangle + Y\langle Z, X\rangle - Z\langle X, Y\rangle = \\ & \qquad 2\langle \nabla_X Y, Z\rangle + \big\langle Y, [X, Z]\big\rangle + \big\langle Z, [Y, X]\big\rangle - \big\langle X, [Z, Y]\big\rangle.\end{aligned}$$

Solving for $\langle \nabla_X Y, Z\rangle$, one gets (1.4.6). Now suppose ∇^1 and ∇^2 are two connections satisfying (1.4.6). Since the right-hand side of (1.4.6) does not depend on the connection, one has that $\big\langle \nabla^1_X Y - \nabla^2_X Y, Z\big\rangle = 0$ for all X, Y, Z. This is possible only if $\nabla^1_X Y = \nabla^2_X Y$ for all X and Y. Hence $\nabla^1 = \nabla^2$. □

Hereafter, we will tacitly assume that the connection ∇ in consideration is the Levi-Civita connection, unless indicated otherwise.

1.4.4 Sectional curvature

For a manifold M equipped with the Levi-Civita connection, the *sectional curvature* K is defined in terms of the curvature tensor R defined in (1.4.2) as [60, Proposition 8.29]:

$$K\left(X,Y\right) = \frac{\left\langle R\left(X,Y\right)Y,X\right\rangle}{\|X\|^2\,\|Y\|^2 - \langle X,Y\rangle^2}. \tag{1.4.10}$$

The numerator of the right-hand side of (1.4.10) is the so-called *Riemannian curvature tensor* Rm given by:

$$Rm\left(X,Y,Z,W\right) = \left\langle R\left(X,Y\right)Z,W\right\rangle. \tag{1.4.11}$$

Example 6 ([60, Theorem 8.34])**.**

- *The Euclidean space $\mathbb{R}^n$ has constant sectional curvature $K = 0$.*
- *The sphere of radius R has constant sectional curvature $K = 1/R^2$.*
- *The hyperbolic space of parameter R has constant sectional curvature $K = -1/R^2$.*

Example 7. *The space $\mathbb{S}^n_{++}$ of symmetric positive definite matrices has non-positive non-constant sectional curvature. The sectional curvatures of $\mathbb{S}^n_{++}$ are contained in the interval $\left[-\frac{1}{2},0\right]$, see [26, Proposition I.1].*

Remark 1. *For a Lie group G with a left-invariant metric g in the Lie algebra, the sectional curvature of G at any point is determined by the curvature at the identity $e \in G$. Therefore, it is enough to describe the curvatures in the Lie algebra $\mathfrak{g} := \mathrm{T}_e G$. The formula for the curvature is the same whether G is equipped with a right-invariant Riemannian metric or a left-invariant one. For Lie groups equipped with a bi-invariant metric, the sectional curvatures are nonnegative in all directions, see [6, Section IV.2].*

Remark 2. *There exist no simply connected compact manifolds that admit a metric of non-positive sectional curvature, see [60, Corollary 12.11].*

Remark 3 (Cartan-Hadamard theorem)**.** *All complete, simply connected manifolds of non-positive sectional curvature are diffeomorphic to $\mathbb{R}^n$, see [60, Theorem 12.8].*

1.4.5 Geodesics

Geodesics are the generalisation of straight lines in Euclidean space to Riemannian manifolds. A curve $\gamma : [p, q] \to M$ is called a *geodesic* if it has zero acceleration, i.e., if it has constant velocity. Geodesics are locally length-minimizing curves. The geodesic $\gamma(t)$ with $\gamma(0) = p$ and $\dot{\gamma}(0) = v$ is denoted as $\gamma(t) = \exp_p(tv)$. The velocity vector $\dot{\gamma}(t)$ is an example of a vector field along a curve. For the interpretation of acceleration of a curve on a manifold, connections come into play as a coordinate-independent way of differentiating vector fields along curves. For every smooth curve γ, the *acceleration of* γ is defined as the vector field $D_t\dot{\gamma}(t)$ along γ. Finally, the curve γ is called a geodesic if along $\gamma(t)$,

$$D_t\dot{\gamma}(t) = 0, \tag{1.4.12}$$

which is equivalent to saying that a geodesic can be characterised as a curve whose velocity vector field is parallel along the curve.

1.4.6 Non-expansive systems on Riemannian manifolds

When investigating the stability behaviour of numerical integrators, one has first to make sure that the solutions of the ODE exhibit a certain stable behaviour and then expect the integrator to mimic it. In our work on the B-stability of numerical integrators on Riemannian manifolds, we assume the ODE system to be non-expansive and then investigate the B-stability of the integrators applied to such systems. In this paragraph, we discuss in more detail non-expansive systems and the tools employed in their definition.

The *t-flow* of a vector field $X \in \mathcal{X}\left((M, g)\right)$, denoted as $\exp(tX)$, is the diffeomorphism on M, $p \mapsto y(t)$ where $\dot{y} = X(y)$, $y(0) = p$, and its domain of definition may be t-dependent. A set $\mathcal{U} \subseteq M$ is *geodesically convex* if, for each $p, q \in \mathcal{U}$, there is a unique minimising geodesic segment from p to q contained entirely in $\mathcal{U}$. A vector field X is *forward complete on* $\mathcal{U}$ if for every $p \in \mathcal{U}$, $\exp(tX)\,p$ is defined for all $t \geq 0$. If for every $(t, p) \in [0, \infty) \times \mathcal{U}$ it holds that $\exp(tX)\,p \in \mathcal{U}$, we say that $\mathcal{U}$ is *forward X-invariant.*

We say that the vector field $X \in \mathcal{X}\left((M, g)\right)$ satisfies a *monotonicity condition* on the set $\mathcal{U} \subseteq (M, g)$ with constant $\nu \in \mathbb{R}$ if for every $x \in \mathcal{U}$ and $v_x \in T_xM$, it holds that

$$\langle \nabla_{v_x} X, v_x \rangle \leq \nu \left\| v_x \right\|^2, \tag{1.4.13}$$

where ∇ is the Levi-Civita connection. We call *non-expansive system* a quadruple $(\mathcal{U}, X, g, \nu)$ where $X \in \mathcal{X}\left((M, g)\right)$ is forward complete on an open, geodesically convex set $\mathcal{U} \subseteq (M, g)$, $\mathcal{U}$ is forward X-invariant, and X satisfies the monotonicity condition (1.4.13) on $\mathcal{U}$ with $\nu \leq 0$. It is important to note here that the non-expansivity of a system depends on the choice of the inner product norm.

1.4.7 Families of curves and the Jacobi equation

Let (M, g) be a Riemannian manifold and $I, J \subseteq \mathbb{R}$ two given intervals. A continuous map $\Gamma : J \times I \to M$ is called a *one-parameter family of curves* and it defines two collections of curves in M: the curves $\Gamma_s(t) = \Gamma(s,t)$ defined for $t \in I$ by holding s constant, and the curves $\Gamma^{(t)}(s) = \Gamma(s,t)$ defined for $s \in J$ by holding t constant. If Γ is smooth, the velocity vectors of the two groups of curves are denoted as

$$\partial_t \Gamma(s,t) = \left(\Gamma_s\right)'(t) \in T_{\Gamma(s,t)}M, \tag{1.4.14}$$

$$\partial_s \Gamma(s,t) = \Gamma^{(t)\prime}(s) \in T_{\Gamma(s,t)}M. \tag{1.4.15}$$

(1.4.14) and (1.4.15) are examples of *a vector field along* Γ, i.e., a continuous map $V : J \times I \to TM$ such that $V(s,t) \in T_{\Gamma(s,t)}M$ for each (s,t).

$\Gamma : J \times I \to M$ with J some open interval and $I = [a,b]$ is called an *admissible family of curves* if it is smooth on each rectangle $J \times [a_{i-1}, a_i]$ for a finite partition $a = a_0 < \ldots < a_k = b$, and $\Gamma_s(t) = \Gamma(s,t)$ is an admissible curve for each $s \in J$. If $\gamma : [a,b] \to M$ is a given admissible curve, *a variation of* γ is an admissible family of curves $\Gamma : J \times [a,b] \to M$ such that J is an open interval containing 0 and $\Gamma_0 = \gamma$. If Γ is a variation of γ, the piecewise smooth vector field $V(t) = \partial_s \Gamma(0,t)$ along γ is called *the variation field of* Γ. These are all useful instruments one can use to investigate the stability behaviour of the numerical integrators whose numerical approximation lies on the geodesics of the manifold. To study the effect of curvature on nearby geodesics, it is useful to focus on variations through geodesics.

Let $\gamma : I \to M$ be a geodesic, and $\Gamma : K \times I \to M$ be a variation of γ, where $I, K \subseteq \mathbb{R}$ are intervals. Γ is called a *variation through geodesics* if each of the curves $\Gamma_s(t) = \Gamma(s,t)$ is also a geodesic. There exists an equation, the so-called *Jacobi equation*, that must be satisfied by the variation field of a variation through geodesics.

Theorem 1.4 ([60, Theorem 10.1]). *Let (M, g) be a Riemannian manifold, let γ be a geodesic in M, and let J be a vector field along γ. If J is the variation field of a variation through geodesics, then J satisfies the following equation, called the Jacobi equation:*

$$D_t^2 J + R\left(J, \gamma'\right)\gamma' = 0, \tag{1.4.16}$$

where R is the curvature tensor from (1.4.2). *A smooth vector field along a geodesic that satisfies the Jacobi equation is called a Jacobi field.*

1.5 Machine learning in computational mechanics

In the past few years, machine learning (ML) techniques have become widely used in the framework of computational mechanics, [12, 23, 27, 30, 32, 38, 51, 69–72, 75, 80, 95, 98, 100]. These are techniques that rely on large amounts of available data and on known physical principles. The treatment we provide here is heavily based on the introductory course [54].

Two important types of learning are *unsupervised* and *supervised* learning. Unsupervised ML models are given unlabelled data and try to find inherent structures or repeating patterns in the data. In the supervised learning setting, the data set under consideration is labelled, i.e., next to each point in the data set there is a label or target associated with it. Consider for example an image classification task where each image has been labelled under a certain category. A supervised learning technique classifies the images into the given categories by comparing its predictions with the true labels. In this thesis, we have worked with supervised learning problems.

Most of the tasks generally approached with machine learning fall under one of the following: *regression*, *classification*, or *clustering* tasks. Given some labelled data points, regression models involve the approximation of an unknown target function, typically continuous, that defines the relationship between independent variables and a dependent variable, or outcome, used to predict the label value for any new set of input features. Classification models differ from regression ones because their output has a discrete form. They take as input a set of labelled data, where each label corresponds to a category, and output the predicted category of new unlabelled instances of data. Both regression and classification are supervised learning tasks. Clustering is an unsupervised learning task which aggregates the data based on the underlying similarities.

Logistic regression (LogR), introduced in [8], is a ML technique widely used in areas such as biology, medicine, psychology, finance and economics. It is one of the most used algorithms for classification tasks, thanks to its simplicity, efficiency and interpretability, [44, 50, 78]. Support Vector Machine (SVM) [10], is another ML model and is used for binary classification of data applied in many fields [21]. Decision Trees (DTs), introduced in [79], are a popular technique for classification and regression. For applications of DTs see [91, 105, 110]. These are all supervised techniques and have been shown to perform well when used as time series classification methods to detect anomalies in subsea engineering, see Chapter 6.

1.5.1 Neural networks

For supervised learning tasks, neural networks represent another effective approach. They are used for regression, classification, and clustering. A *neural network* is a parametric function $f_{\boldsymbol{\rho}} : \mathcal{I} \to \mathcal{O}$ with parameters $\boldsymbol{\rho} = (\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\ell)$ given as a composition of multiple transformations,

$$f_{\boldsymbol{\rho}} := f_{\boldsymbol{\rho}_\ell} \circ \cdots \circ f_{\boldsymbol{\rho}_j} \circ \cdots \circ f_{\boldsymbol{\rho}_1},$$

where each $f_{\boldsymbol{\rho}_j}$ represents the j-th layer of nodes, or *neurons*, of the network, with $j = 1, \ldots, \ell$, and ℓ being the number of layers.

The majority of ML algorithms are composed of a cost function, an optimisation procedure, and a parametrised model. We discuss these components via the following example of a neural network type.

Example 8 (The multi-layer perceptron)**.** *A multi-layer perceptron (MLP) is a fully connected feed-forward neural network, i.e., every neuron from the previous layer is connected with each neuron of the next layer. The j-th layer of an MLP is defined as*

$$f_j^{MLP}(\mathbf{x}) = \sigma\left(\mathbf{A}_j \mathbf{x} + \mathbf{b}_j\right) \in \mathbb{R}^{n_j}, \tag{1.5.1}$$

where σ is the so-called activation function, $\mathbf{x} \in \mathbb{R}^{n_{j-1}}$, 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., $\boldsymbol{\rho} = \left\{\mathbf{A}_j, \mathbf{b}_j\right\}_{j=1}^{\ell}$.

A neural network with at least one hidden layer containing a sufficient number of neurons, and a non-linear activation function can approximate any continuous multi-input/multi-output function with arbitrary precision. This is known as the Universal Approximation Theorem [28, 68, 90].

The activation function σ is a continuous nonlinear scalar function, which acts component-wise on vectors. Different types of activation functions exist, with the most common ones being the hyperbolic tangent, the sigmoid function, and the rectified linear unit (ReLU), which can be written respectively as

$$\sigma(z) = \tanh(z), \qquad \sigma(z) = \frac{1}{1+e^{-z}}, \qquad \sigma(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z > 0 \end{cases}.$$

The weights $\boldsymbol{\rho}$ are chosen such that $f_{\boldsymbol{\rho}}$ approximates accurately enough a map of interest f. Usually, this is done by minimising a suitable *loss function* $\mathrm{Loss}(\boldsymbol{\rho})$. In the context of supervised learning, 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 measures 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 process of minimising Loss($\boldsymbol{\rho}$) with respect to $\boldsymbol{\rho}$ is known as the *training* of the network and is usually done with *gradient descent (GD)*:

$$\boldsymbol{\rho}^{(k)} \mapsto \boldsymbol{\rho}^{(k)} - \eta\nabla\text{Loss}\left(\boldsymbol{\rho}^{(k)}\right) =: \boldsymbol{\rho}^{(k+1)}, \tag{1.5.2}$$

also known as *steepest descent*. The scalar value η is known as the *learning rate*. $\nabla\text{Loss}\left(\boldsymbol{\rho}^{(k)}\right)$ is a sum over the training data of individual partial derivatives. The computation of the partial derivatives of the loss function with respect to each network parameter is done through the so-called backpropagation algorithm, which relies on an efficient use of the chain rule for differentiation, see e.g. [46, Section 5], [54, Section 3.4]. A drawback of (1.5.2) is its sensitivity to the choice of η. If η is too large, the algorithm starts to oscillate or even fails to converge at all, and if η is too small, one has a very low convergence rate. Another drawback is the high computational cost due to two reasons: the large number of data points and the large dimensions of the weight matrices $\mathbf{A}_j$. A much cheaper alternative to gradient descent is *stochastic gradient descent (SGD)*. With SGD the gradient at each iteration in (1.5.2) is not computed using all data points, but rather using randomly chosen subsets $\mathcal{B} \subset \Omega$ of the shuffled data, called *batches*, of cardinality $B = |\mathcal{B}|$. It is called *stochastic* because the gradient of a partition of the data set might point in a significantly different direction than the gradient computed for the whole data set. SGD often shows better convergence properties than GD, especially when combined with an adaptive learning rate. Schemes like GD and SGD are widely used because they require the knowledge only of the first-order derivatives of f, i.e., $\nabla f(x)$. To obtain a faster convergence rate, several modifications of these methods, the so-called accelerated gradient descent methods, have been proposed. Among the most well known of these are Polyak's heavy ball method [92], also known as the classical momentum method, Nesterov's accelerated gradient method [82], and the Adam method [53]. We have used the latter both in Chapter 5 and Chapter 6. Methods that involve the evaluation of the Hessian or an approximation of it are less popular due to the high computational cost and time, and memory requirements. Examples of such methods are Newton's method, the quasi-Newton methods, the trust region methods, see e.g. [84, Sections 3,4,6].

Once the training procedure is complete, the model's accuracy is measured in predicting the correct output for new inputs included in the test set that are unseen during training.

We have used neural networks in Chapter 5 to approximate configurations of beams. The following is an example of such a task.

Example 9 (Euler's elastica)**.** *We consider the Euler's elastica [104], an inextensible beam model, and starting from a data set of solutions of the discretised static equilibria, we train the neural networks to produce solutions for unseen boundary conditions. The data set consists of 1000 trajectories, discretised in 50 intervals, with* $90\% - 10\%$ *splitting into training and test set. Figure 1.3 shows the comparison between the true and the predicted trajectories over the test set.* q_x *and* q_y *denote the components of the positions. For presentation purposes, only 4 randomly selected trajectories are considered.*

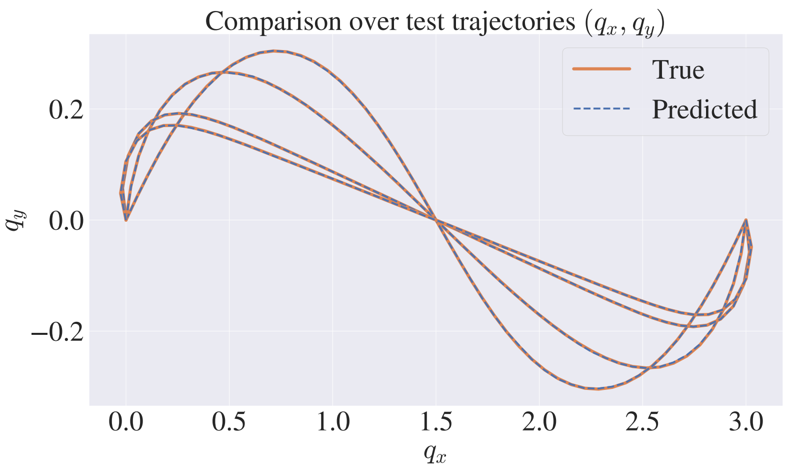

Figure 1.3: Comparison over test trajectories for the positions $\left(q_x, q_y\right)$. These results are obtained with the MLP architecture from Example 8, with ReLU as activation function, 2 layers, 824 hidden nodes, learning rate $1.151 \cdot 10^{-3}$, and batch size 64. This combination of hyperparameters yields a training error equal to $3.668 \cdot 10^{-7}$ and a test error equal to $4.518 \cdot 10^{-7}$.

1.5.2 Advanced architectures and classes of neural networks

Two more advanced neural network architectures are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

CNNs were introduced in [57] and were originally designed for image processing [56, 58]. Their name comes from the fact that they use the convolution operation $(*)$ instead of general matrix multiplication in at least one of their layers. The input they take comes in the form of a matrix, or grid, allowing to account for 2D grids of pixels in an image. Although CNNs were specifically introduced to work with image data, they have reached state of the art results also in other fields. In particular, they have proved to be effective at processing data in the form of time series, thought of as a 1D grid taking samples at regular time intervals, making them among the most successful deep learning architectures for time series processing [7,33,59]. We have used CNNs for this purpose in Chapter 6.

RNNs, introduced in [97], are neural networks designed for processing sequential data. They are networks with internal loops that allow information to persist and propagate over time. A RNN can be thought of as multiple instances of the same neural network where each network propagates information to the successive one. RNNs are widely applied in various fields. These include speech recognition, language modelling and generation, and machine translation, see e.g. [36,56,106]. For a more thorough treatment of CNNs and RNNs, we refer to [35, Chapters 10-11].

In cases when neural networks are applied to physical problems and trained only based on observations, they do not have knowledge of the physical information of the system under consideration. This prevents them from respecting the underlying physical laws. This issue was addressed in [55], and was recently revived in [95] in the framework of Physics-Informed Neural Networks (PINNs) and developed further in many directions, see e.g. [27,54,100]. PINNs incorporate physical laws and principles, often expressed as ODEs or PDEs, into the learning process. This is done by augmenting the loss function used to train the neural network with a term enforcing the differential equation on a set of collocation points inside the domain.

Neural networks have also been applied to learn physical models of classical mechanics based on data. For Hamiltonian systems, Hamiltonian Neural Networks (HNNs) were proposed in [37] to learn the energy function from the position and momentum data of trajectories. In [18], new approaches for the accurate approximation of the Hamiltonian function of constrained mechanical systems were proposed. In [25], Lagrangian Neural Networks (LNNs) were proposed. These are neural networks that can parameterise arbitrary Lagrangians without requiring canonical coordinates. In [87] an approach that learns an inverse modified Hamiltonian using kernel-based methods is presented. A variational version of the approach in [87] that learns inverse modified Lagrangian functions is presented in [86].

1.6 Summary of papers

The layout, bibliography, and typography of the included papers have been unified, a few typos have been corrected, and missing punctuation marks have been added. To fit the B5 format of this thesis, figures have been allowed to float freely and a few equations have been reformatted. No other substantial modifications have been made to the papers.

Paper 1: Lie group integrators for mechanical systems

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

International Journal of Computer Mathematics, Vol.99, No.1, 2022, pp. 58-88.

In this paper, we consider Lie group integrators with a particular focus on problems from mechanics. We present the Runge-Kutta-Munthe-Kaas(RKMK) and the commutator-free classes of Lie group integrators and discuss ways of adapting the integration step size in time. We discuss Hamiltonian systems on Lie groups and consider three equivalent formulations of the heavy top as an application example to which symplectic Lie group integrators are applied. The RKMK and the commutator-free integrators are tested on two more complex beam and multi-body inspired problems, the chain of pendula and a system of two quadrotors transporting a mass point. The Lie groups and manifolds of interest are $SO(n), SE(n), n = 2,3$, S^2, and TS^2.

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

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

In: M. Ehrhardt, M. Günther (eds) Progress in Industrial Mathematics at ECMI 2021. ECMI 2021. Mathematics in Industry(), Vol 39. Springer, Cham., 2022, pp. 297-304.

This paper is an extension of the previous one. We provide a brief overview of the RKMK methods with a particular focus on adaptive time-stepping techniques. We consider again the chain of N connected $3D$ pendulums, whose dynamics evolves on $\left(TS^2\right)^N$. We introduce the 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. The final part shows numerical experiments comparing constant and variable step size methods applied to the N-fold pendulum.

Paper 3: B-stability of numerical integrators on Riemannian manifolds

Martin Arnold, Elena Celledoni, Ergys Çokaj,
Brynjulf Owren, and Denise Tumiotto

Journal of Computational Dynamics Vol. 11, No. 1, 2024, pp. 92-107.

In this paper, the notion of B-stability of numerical integrators in Euclidean spaces proposed in [13] is generalised for the first time to numerical integrators on Riemannian manifolds. We introduce non-expansive systems on such manifolds and define the B-stability of integrators in terms of the Riemannian distance function. We consider a geodesic version of the Implicit Euler scheme (GIE) and prove that it is B-stable on Riemannian manifolds with non-positive sectional curvature. We provide numerical evidence that the GIE method is expansive when applied to a certain non-expansive vector field on the 2-sphere, and that it can become multivalued for large enough step sizes, which is in contrast to what has been proved in Euclidean spaces. We conclude by deriving a new improved global error estimate for general Lie group integrators.

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, and Martina Stavole

Submitted to *Multibody System Dynamics*.

In this paper, we present a *discrete* and a *continuous* neural network based approach for the approximation of Euler's elastica, a classical model of flexible slender structures, relevant in many industrial applications. We start from a data set of solutions of the discretised static equilibria and then we train the neural networks to produce solutions for unseen boundary conditions. The discrete approach learns discrete solutions from the discrete data, while the continuous approach computes an arc length parametrisation of the beam configuration. We present numerical evidence that the proposed approaches effectively approximate configurations of the elastica for a range of different boundary conditions.

Paper 5: Supervised Time Series Classification for Anomaly Detection in Subsea Engineering

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

Accepted for publication by the *Journal of Computational Dynamics*.

In this work, we use supervised machine learning models to perform binary classification on a simulated data set based on a physical system with two states: Intact and Broken. We discuss 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 proceed with the presentation of four methods, Logistic Regression (LogR), Decision Trees (DTs), Support Vector Machines (SVMs), and Convolutional Neural Networks (CNNs), and test them on our data set. The experimental results suggest that machine learning techniques are advantageous as a tool in decision making.

Bibliography

[1] S. S. Antmann, *Nonlinear Problems in Elasticity, (2nd edn.)*, Springer, 2005.

[2] M. Arnold and O. Brüls, *Convergence of the generalized-α scheme for constrained mechanical systems*, Multibody System Dynamics **18** (2007), no. 2, 185–202.

[3] M. Arnold, O. Brüls, and A. Cardona, *Error analysis of generalized-α Lie group time integration methods for constrained mechanical systems*, Numerische Mathematik **129** (2015), no. 1, 149–179.

[4] M. Arnold, O. Brüls, E. Celledoni, S. Chandrashekara, E. Çokaj, M. Debeurre, V. Dörlich, G. Jelenić, A. Leone, S. Leyendecker, J. Linn, D. Manfredo, B. Owren, I. Patil, M. Stavole, O. Thomas, J. Tomec, D. Tumiotto, and D. Zupan, *Report on methods (benchmark test), Deliverable 3.3*, Tech. report, EU Horizon 2020 THREAD Project. Grant agreement No. 860124, 2023.

[5] M. Arnold, E. Celledoni, E. Çokaj, B. Owren, and D. Tumiotto, *B-stability of numerical integrators on Riemannian manifolds*, Journal of Computational Dynamics **11** (2024), no. 1, 92–107.

[6] V. I. Arnold, *Mathematical methods of classical mechanics*, GTM 60, Second ed., Springer-Verlag, 1989.

[7] 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** (2023), no. 5, 28–44.

[8] J. Berkson, *Application of the logistic function to bio-assay*, Journal of the American Statistical Association **39** (1944), no. 227, 357–365.

[9] G. Bogfjellmo and H. Marthinsen, *High-order symplectic partitioned Lie group methods*, Foundations of Computational Mathematics **16** (2016), no. 2, 493–530.

[10] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the Fifth Annual Workshop on Computational Learning Theory (New York, NY, USA), COLT '92, Association for Computing Machinery, 1992, p. 144–152.

[11] O. Bruls and A. Cardona, *On the Use of Lie Group Time Integrators in Multibody dynamics*, Journal of Computational Nonlinear Dynamics **5** (2010), no. 3, 031002.

[12] S. L. Brunton and J. N. Kutz, *Machine learning for partial differential equations*, 2023, arXiv:2303.17078.

[13] J. C. Butcher, *A stability property of implicit Runge-Kutta methods*, BIT Numerical Mathematics **15** (1975), 358–361.

[14] E. Çokaj, H. S. Gustad, A. Leone, P. T. Moe, and L. Moldestad, *Supervised time series classification for anomaly detection in subsea engineering*, 2024, arXiv:2403.08013.

[15] E. Celledoni, E. Çokaj, A. Leone, S. Leyendecker, D. Murari, B. Owren, R. T. S. M. de Almagro, and M. Stavole, *Neural networks for the approximation of Euler's elastica*, 2023, arXiv:2312.00644.

[16] E. Celledoni, E. Çokaj, A. Leone, D. Murari, and B. Owren, *Dynamics of the N-fold pendulum in the framework of Lie group integrators*, Progress in Industrial Mathematics at ECMI 2021 (Cham) (Matthias Ehrhardt and Michael Günther, eds.), Springer International Publishing, 2022, pp. 297–304.

[17] ———, *Lie group integrators for mechanical systems*, International Journal of Computer Mathematics **99** (2022), no. 1, 58–88.

[18] E. Celledoni, A. Leone, D. Murari, and B. Owren, *Learning hamiltonians of constrained mechanical systems*, Journal of Computational and Applied Mathematics **417** (2023), 114608.

[19] E. Celledoni, H. Marthinsen, and B. Owren, *An introduction to Lie group integrators—basics, new developments and applications*, Journal of Computational Physics **257** (2014), 1040–1061.

[20] E. Celledoni and B. Owren, *Lie group methods for rigid body dynamics and time integration on manifolds*, Computer Methods in Applied Mechanics and Engineering **192** (2003), no. 3-4, 421–438.

[21] 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** (2020), 189–215.

[22] 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 Humanoid Robotics, Cancun, Mexico, 15–17 November 2016 (2016), 826–833.

[23] S. Chevalier, J. Stiasny, and S. Chatzivasileiadis, *Accelerating Dynamical System Simulations with Contracting and Physics-Projected Neural-Newton Solvers*, Learning for Dynamics and Control Conference (PMLR), 2022, pp. 803–816.

[24] E. Cosserat and F. Cosserat, *Theorie des corps deformables*, Hermann, Paris, 1909.

[25] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, *Lagrangian neural networks*, 2020, arXiv preprint arXiv:2003.04630.

[26] C. Criscitiello and N. Boumal, *An accelerated first-order method for non-convex optimization on manifolds*, Foundations of Computational Mathematics **23** (2023), no. 4, 1433–1509.

[27] 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** (2022), no. 3, 88.

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

[29] M. Dahlby, B. Owren, and T. Yaguchi, *Preserving multiple first integrals by discrete gradients*, Journal of Physics. A. Mathematical and Theoretical **44** (2011), no. 30, 305205, 14.

[30] 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** (2022), no. 6, 063107.

[31] F. Demoures, F. Gay-Balmaz, S. Leyendecker, S. Ober-Blöbaum, T. S. Ratiu, and Y. Weinand, *Discrete variational Lie group formulation of geometrically exact beam dynamics*, Numerische Mathematik **130** (2015), no. 1, 73–123.

[32] 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** (2023), no. 4, 043128.

[33] 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** (2019), no. 4, 917–963.

[34] O. Gonzalez, *Time integration and discrete Hamiltonian systems*, Journal of Nonlinear Science **6** (1996), 449–467.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[36] A. Graves, A. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, 2013 IEEE international conference on acoustics, speech and signal processing, Ieee, 2013, pp. 6645–6649.

[37] S. Greydanus, M. Dzamba, and J. Yosinski, *Hamiltonian neural networks*, Advances in neural information processing systems **32** (2019), 1–11.

[38] 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** (2023), no. 5, A2356–A2381.

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

[40] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations I, Nonstiff problems*, Second revised edition ed., Springer-Verlag, 1993.

[41] J. Hall and M. Leok, *Lie group spectral variational integrators*, Foundations of Computational Mathematics **17** (2017), no. 1, 199–257.

[42] S. Hante and M. Arnold, *RATTLie: A variational Lie group integration scheme for constrained mechanical systems*, Journal of Computational and Applied Mathematics **387** (2021), 112492.

[43] S. Hante, D. Tumiotto, and M. Arnold, *A Lie group variational integration approach to the full discretization of a constrained geometrically exact Cosserat beam model*, Multibody System Dynamics **54** (2022), 97–123.

[44] F. E. Harrell, *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, vol. 608, Springer, 2001.

[45] F. Hausdorff, *Die symbolische Exponentialformel in der Gruppentheorie*, Leipziger Ber. **58** (1906), 19–48.

[46] C. F. Higham and D. J. Higham, *Deep learning: An introduction for applied mathematicians*, Siam review **61** (2019), no. 4, 860–891.

[47] 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** (1977), no. 3, 283–292.

[48] S. Holzinger and J. Gerstmayr, *Time integration of rigid bodies modelled with three rotation parameters*, Multibody System Dynamics (2021), 1–34.

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

[50] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, vol. 398, John Wiley & Sons, 2013.

[51] 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** (2022), no. 2, 025002.

[52] F. Kang and S. Zai-jiu, *Volume-preserving algorithmms for source-free dynamical systems*, Numer. Math. **71** (1995), no. 4, 451–463.

[53] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, International Conference on Learning Representations (ICLR) (San Diega, CA, USA), 2015.

[54] S. Kollmannsberger, D. D'Angella, M. Jokeit, and L. Herrmann, *Deep learning in computational mechanics*, Springer, 2021.

[55] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks **9** (1998), no. 5, 987–1000.

[56] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature **521** (2015), no. 7553, 436–444.

[57] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation **1** (1989), no. 4, 541–551.

[58] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[59] D. Lee, S. Malacarne, and E. Aune, *Vector quantized time series generation with a bidirectional prior model*, Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, eds.), Proceedings of Machine Learning Research, vol. 206, PMLR, 4 2023, pp. 7665–7693.

[60] J. M. Lee, *Introduction to Riemannian manifolds*, Graduate Texts in Mathematics, vol. 176, Springer, Cham, 2018.

[61] T. Lee, M. Leok, and N. H. McClamroch, *Lie group variational integrators for the full body problem*, Computer Methods in Applied Mechanics and Engineering **196** (2007), no. 29-30, 2907–2924.

[62] ———, *Global formulations of Lagrangian and Hamiltonian dynamics on manifolds*, Interaction of Mechanics and Mathematics, Springer, Cham, 2018, A geometric approach to modeling and analysis.

[63] B. Leimkuhler and S. Reich, *Simulating hamiltonian dynamics*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2005.

[64] T. Leitz, R. T. S. M. de Almagro, and S. Leyendecker, *Multisymplectic Galerkin Lie group variational integrators for geometrically exact beam dynamics based on unit dual quaternion interpolation – no shear locking*, Computer Methods in Applied Mechanics and Engineering **374** (2021), 113475.

[65] T. Leitz and S. Leyendecker, *Galerkin Lie-group variational integrators based on unit quaternion interpolation*, Computer Methods in Applied Mechanics and Engineering **338** (2018), 333–361.

[66] D. Lewis and J. C. Simo, *Conserving algorithms for the dynamics of Hamiltonian systems of Lie groups*, Journal of Nonlinear Science **4** (1994), 253–299.

[67] S. Leyendecker, *Advanced geometric integrators, Deliverable 3.1*, Tech. report, EU Horizon 2020 THREAD Project. Grant agreement No. 860124, 2020.

[68] Q. Li, T. Lin, and Z. Shen, *Deep learning via dynamical systems: An approximation perspective*, Journal of the European Mathematical Society **25** (2022), no. 5, 1671–1709.

[69] Y. Li, Z. Zhou, and S. Ying, *Delisa: Deep learning based iteration scheme approximation for solving PDEs*, Journal of Computational Physics **451** (2022), 110884.

[70] Y. Liu, J. N. Kutz, and S. L. Brunton, *Hierarchical deep learning of multiscale differential equation time-steppers*, 2020, arXiv:2008.09768.

[71] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*, Nature machine intelligence **3** (2021), no. 3, 218–229.

[72] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *Deepxde: A deep learning library for solving differential equations*, SIAM review **63** (2021), no. 1, 208–228.

[73] J. E. Marsden and T. S. Ratiu, *Introduction to mechanics and symmetry*, Springer-Verlag, 1994.

[74] J. E. Marsden and M. West, *Discrete mechanics and variational integrators*, Acta Numerica **10** (2001), 357–514.

[75] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas, *Hamiltonian neural networks for solving equations of motion*, Physical Review E **105** (2022), no. 6, 065305.

[76] R. I. McLachlan and G. R. W. Quispel, *Splitting methods*, Acta Numerica **11** (2002), 341–434.

[77] R. I. McLachlan, G. R. W. Quispel, and N. Robidoux, *Geometric integration using discrete gradients*, Philosophical Transactions of the Royal Society A **357** (1999), 1021–1046.

[78] S. Menard, *Logistic regression: From introductory to advanced concepts and applications*, Sage, 2010.

[79] J. N. Morgan and J. A. Sonquist, *Problems in the analysis of survey data, and a proposal*, Journal of the American statistical association **58** (1963), no. 302, 415–434.

[80] D. Mortari, H. Johnston, and L. Smith, *High accuracy least-squares solutions of nonlinear differential equations*, Journal of computational and applied mathematics **352** (2019), 293–307.

[81] J. Moser and A. P. Veselov, *Discrete versions of some classical integrable systems and factorization of matrix polynomials*, Communications in Mathematical Physics **139** (1991), no. 2, 217–243.

[82] Y. Nesterov, *A method of solving a convex programming problem with convergence rate o (1/k** 2)*, Doklady Akademii Nauk SSSR **269** (1983), no. 3, 543.

[83] N. M. Newmark, *A method of computation for structural dynamics*, Journal of the Engineering Mechanics Division **85** (1959), no. 3, 67–94.

[84] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer, 1999.

[85] R. A. Norton and G. R. W. Quispel, *Discrete gradient methods for preserving a first integral of an ordinary differential equation*, Discrete and Continuous Dynamical Systems **34** (2014), no. 3, 1147–1170.

[86] S. Ober-Blöbaum and C. Offen, *Variational learning of Euler–Lagrange dynamics from data*, Journal of Computational and Applied Mathematics **421** (2023), 114780.

[87] C. Offen and S. Ober-Blöbaum, *Symplectic integration of learned hamiltonian systems*, Chaos: An Interdisciplinary Journal of Nonlinear Science **32** (2022), no. 1, 1–10.

[88] P. J. Olver, *Equivalence, invariants, and symmetry*, Cambridge University Press, 1995.

[89] J. Park and W. Chung, *Geometric integration on Euclidean group with application to articulated multibody systems*, Journal of Computational and Applied Mathematics **21** (2005), no. 5, 850–863.

[90] A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta numerica **8** (1999), 143–195.

[91] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman, *Decision trees: an overview and their use in medicine*, Journal of medical systems **26** (2002), 445–463.

[92] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*, Ussr computational mathematics and mathematical physics **4** (1964), no. 5, 1–17.

[93] G. R. W. Quispel, *Volume-preserving integrators*, Physics Letters A **206** (1995), 26–30.

[94] G. R. W. Quispel and D. I. McLaren, *A new class of energy-preserving numerical integration methods*, Journal of Physics. A. Mathematical and Theoretical **41** (2008), no. 4, 045206, 7.

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

[96] CORDIS EU research result, *Horizon 2020 Joint Training on Numerical Modelling of Highly Flexible Structures for Industrial Applications*, 2019-2024, `https://cordis.europa.eu/project/id/860124/results`.

[97] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature **323** (1986), no. 6088, 533–536.

[98] 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** (2020), 112790.

[99] J. M. Sanz-Serna and M. P. Calvo, *Numerical Hamiltonian Problems*, AMMC 7, Chapman & Hall, 1994.

[100] 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** (2021), 334–356.

[101] J. C. Simo, *A finite strain beam formulation. Part I: The three-dimensional dynamic problem.*, Computer Methods in Applied Mechanics and Engineering **49** (1985), 55–70.

[102] J. C. Simo and L. Vu-Quoc, *A three-dimensional finite-strain rod model. Part II: Computational aspects*, Computer Methods in Applied Mechanics and Engineering **58** (1986), no. 1, 79–116.

[103] ———, *On the dynamics of finite-strain rods undergoing large motions – A geometrically exact approach*, Computer Methods in Applied Mechanics and Engineering **66** (1988), 125–161.

[104] D. A. Singer, *Lectures on elastic curves and rods*, AIP Conference Proceedings (American Institute of Physics), vol. 1002, 2008, pp. 3–32.

[105] Y. Y. Song and L. Ying, *Decision tree methods: applications for classification and prediction*, Shanghai archives of psychiatry **27** (2015), no. 2, 130.

[106] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems **27** (2014), 3104–3112.

[107] THREAD, *D3.2 Implementation of the novel geometric algorithms in open-source simulation code*, 2022, `https://github.com/THREAD-3-2`.

[108] L. V. Tu, *An introduction to manifolds*, Springer, 2011.

[109] V. S. Varadarajan, *Lie groups, Lie algebras, and their representations*, GTM 102, Springer-Verlag, 1984.

[110] A. Venkatasubramaniam, J. Wolfson, N. Mitchell, T. Barnes, M. Jaka, and S. French, *Decision trees in epidemiological research*, Emerging themes in epidemiology **14** (2017), 1–12.

[111] V. Wieloch and M. Arnold, *BDF integrators for constrained mechanical systems on Lie groups*, Journal of Computational and Applied Mathematics **387** (2019), 112517.

[112] H. Xue and A. Zanna, *Explicit volume-preserving splitting methods for polynomial divergence-free vector fields*, BIT Numerical Mathematics **53** (2013), 265–281.

Part I

Lie group integrators for mechanical systems

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

International Journal of Computer Mathematics,

Vol.99, No.1, 2022, pp. 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 $\mathbb{R}^6 \times \left(SO(3) \times \mathfrak{so}(3)\right)^2 \times \left(TS^2\right)^2$ 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, 41]. 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 [46], and the methods based on the Cayley transformation [13, 32], 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 [50] and the symplectic and energy-momentum methods by Lewis and Simo [32]. 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 [38, 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, 30]. 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, 45] 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 [40,41] 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 multi-body 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 [47], [27] [9], [25]. The present work is motivated by applications in modeling and simulation of slender structures like Cosserat rods and beams [50], 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., $g\cdot m = \psi\left(g,m\right)$. Let $\mathfrak{g}$ be the Lie algebra of G, and denote by $\exp : \mathfrak{g} \to G$ the exponential map. We define $\psi_* : \mathfrak{g} \to \mathcal{X}\left(\mathcal{M}\right)$ to be the infinitesimal generator of the action, i.e.,

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

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

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

This is the original tool [41] 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 $\mathcal{X}\left(\mathcal{M}\right)$ was introduced assuming that for every $m \in \mathcal{M}$,

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

Then, for any vector field $F \in \mathcal{X}\left(\mathcal{M}\right)$ 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

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

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

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

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

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

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

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. $F_p\big|_p = F|_p$
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. [33].

Example 10. *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

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

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

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

Following [35], 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* $\langle \mu, \xi \rangle = \mu^\top \xi$. *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*

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

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*

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

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

$$y_{n+1} = \exp\left(hF_{y_n}\right) y_n$$

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

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

The transformed differential equation for $\sigma(t)$ makes use of the derivative of the exponential mapping, the reader should consult [41] for details about the derivation, we give the final result

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

The map $v \mapsto \mathrm{dexp}_u(v)$ 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 $\mathrm{ad}_u(v) = [u, v]$ and its powers $\mathrm{ad}_u^2 v = \left[u, [u, v]\right]$ etc, one can write

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

and the inverse is

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

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 $\mathrm{dexp}_u^{-1}(v)$ 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 dexp_u^{-1} with $u = \mathcal{O}(h)$, and thus, the kth iterated commutator $\mathrm{ad}_u^k = \mathcal{O}\left(h^k\right)$. 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 $\mathrm{dexp}_u^{-1} v$, see [5, 42]. We give here one concrete example of an RKMK method proposed in [5]

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

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

$$\mathrm{dexp}_u^{-1}(v) = v - \frac{1}{2} u \times v + \alpha^{-2}\left(1 - \tfrac{\alpha}{2}\cot\tfrac{\alpha}{2}\right) u \times (u \times v)$$

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

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

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.

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

where $f_{n,i} = f\left(Y_{n,i}\right)$. 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 $\mathrm{dexp}_u^{-1}(v)$ in $\mathfrak{se}(3)$

As an alternative to using a truncated version of the infinite series for 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 $\rho = A^\top a$, $\alpha = \|A\|_2$. One can show that for any (A, a) and (B, b) in $\mathfrak{se}(3)$, it holds that

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

where

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

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 $\mathrm{dexp}_{(A,a)}^{-1}(B, b)$ 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 [39] a variety of coordinate mappings for rigid body motions are discussed. For Lie algebras of larger dimension, both the exponential mapping and 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 [46]. 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

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

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

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

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

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

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

$$\begin{aligned} \omega_{(g,\mu)} \left(\left(R_{g*}\xi_1, \delta v_1 \right), \left(R_{g*}\xi_2, \delta v_2 \right) \right) = \langle \delta v_2, \xi_1 \rangle - \langle \delta v_1, \xi_2 \rangle + \\ - \left\langle \mu, [\xi_1, \xi_2] \right\rangle, \quad \xi_1, \xi_2 \in \mathfrak{g}. \end{aligned} \tag{2.3.4}$$

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

$$\Omega_{\left(g, p_g\right)} \left((\delta v_1, \delta \pi_1), (\delta v_2, \delta \pi_2) \right) = \langle \delta \pi_2, \delta v_1 \rangle - \langle \delta \pi_1, \delta v_2 \rangle,$$

by right trivialization.

See [32] 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 $\mathrm{i}_F\left(X\right) := \omega\left(F, X\right)$ for any vector field X. This implies that the map f for such a Hamiltonian vector field gets the form

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

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

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

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

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

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 $\mathfrak{g}^*$ [34]. 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 [36]). 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.

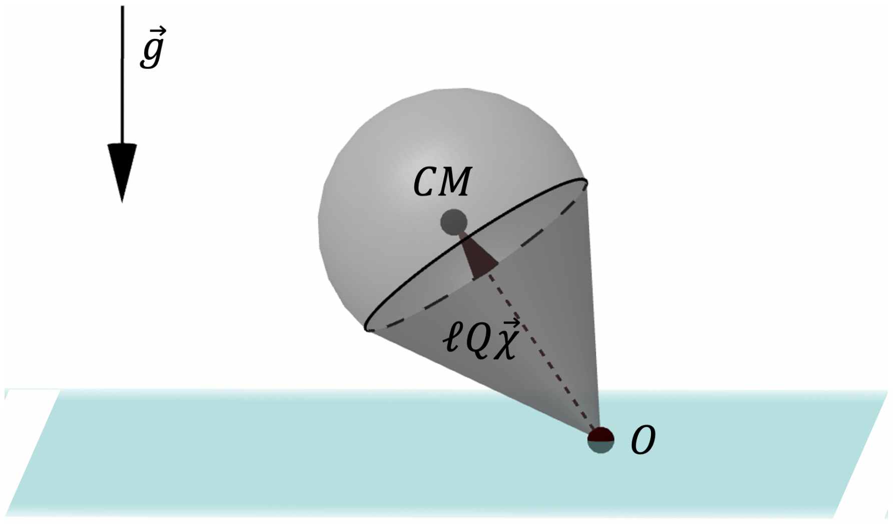

Figure 2.1: Illustration of the heavy top, where CM is the center of mass of the body, O is the fixed point, $\vec{g}$ is the gravitational acceleration vector, and $\ell, Q, \vec{\chi}$ follow the notation introduced in Section 2.3.4.1

2.3.4.1 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

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

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 $\pi = \left(T_eR_Q\right)^*(p) = Q\Pi$. The equations written in the space variables (Q, π) get the form

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

$$\dot{Q} = \hat{\omega}Q, \quad \omega = Q\mathbb{I}^{-1}Q^\top\pi. \tag{2.3.13}$$

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 [32] and [49].

2.3.4.2 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 (2.3.4.1) cannot be reduced to $\mathfrak{so}^*(3)$, nevertheless the heavy top equations are Lie–Poisson on $\mathfrak{se}^*(3)$, [17,48,52].

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

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

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

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

$$\left(g_1, \mathbf{u}_1\right) \cdot \left(g_2, \mathbf{u}_2\right) = \left(g_1g_2, g_1\mathbf{u}_2 + \mathbf{u}_1\right),$$

where g_1g_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

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

Using this expression for $\mathrm{ad}^*_{(\xi,\mathbf{u})}$ with $\left(\xi = \frac{\partial H}{\partial\Pi}, \mathbf{u} = \frac{\partial H}{\partial\Gamma}\right)$, 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].

2.3.4.3 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 $\mathbf{q}$:

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

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

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

and in the spatial momentum variables

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

Similar formulations were considered in [31] for the stability analysis of an underwater vehicle. A similar but different formulation of the heavy top was considered in [4].

2.3.4.4 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 $Q(0) = I$, $\ell = 2$, $M = 15$, $\mathbb{I} = \operatorname{diag}\left(0.234375, 0.46875, 0.234375\right)$, $\pi(0) = \mathbb{I}\left(0, 150, -4.61538\right)$, $\mathcal{X} = \left(0,1,0\right)$ $\Gamma_0 = \left(0,0,-9.81\right)$.

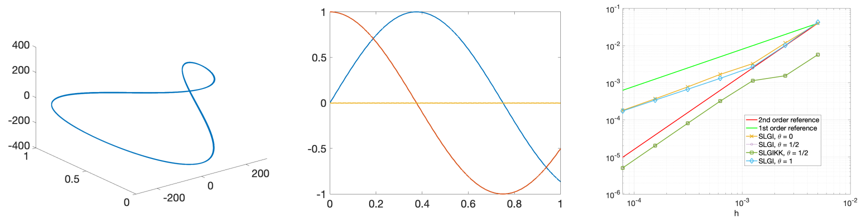

Figure 2.2: Symplectic Lie group integrators integration on the time interval $\left[0,1\right]$. Left: 3D plot of $M\ell Q^{-1}\Gamma_0$. Center: components of $Q\mathcal{X}$. The left and center plots are computed with the same step-size. Right: verification of the order of the methods.

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 [37] 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

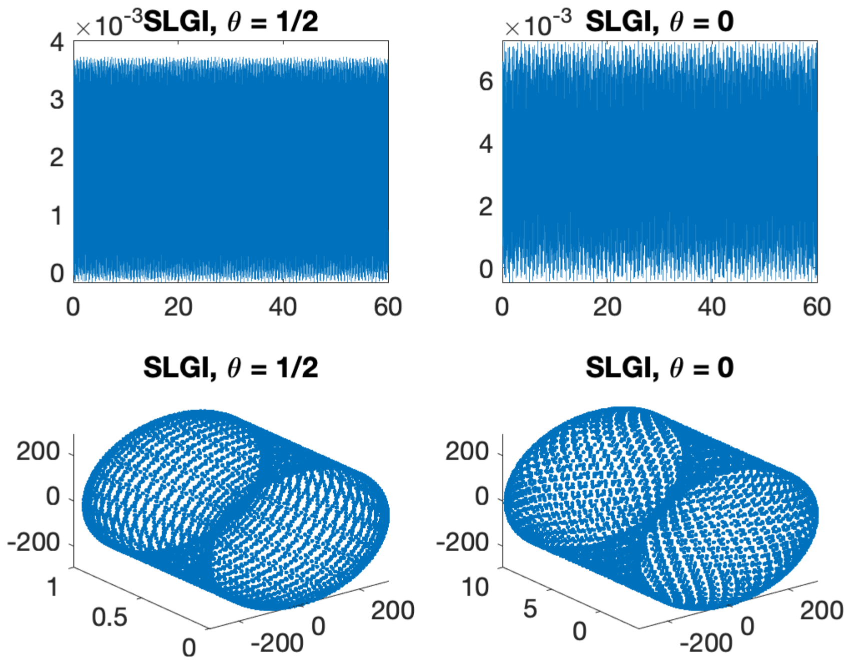

Figure 2.3: Symplectic Lie group integrators, long time integration, $h = 0.01$, 6000 steps. Top: energy error, bottom 3D plot of $M\ell Q^{-1}\Gamma_0$.

approximately equal to some user defined tolerance tol in every step. Suppose 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, $e_{n+1} = Ch_{n+1}^{\tilde{p}+1} + \mathcal{O}\left(h^{\tilde{p}+2}\right)$. Aiming at $e_{n+1} \approx$ tol in every step, one may use a formula of the type

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

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

[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.

```
Given $y_n$, $h_n$, tol
Let $h := h_n$
**repeat**
    Compute $y_{n+1}$, $\tilde{y}_{n+1}$, $e_{n+1}$ from $y_n$, $h$
    Update stepsize $h := \theta\left(\frac{\mathrm{tol}}{e_{n+1}}\right)^{\alpha} h$
    accepted := $e_{n+1} < \mathrm{tol}$
    **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 $\left(a_{ij}, b_i, \tilde{a}_{ij}, \tilde{b}_i\right)$ 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 [44] 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 $e_{n+1} = d\left(y_{n+1}, \tilde{y}_{n+1}\right)$ We give a few examples of embedded pairs.

2.4.2.1 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

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

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

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

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

2.4.2.2 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

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

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.

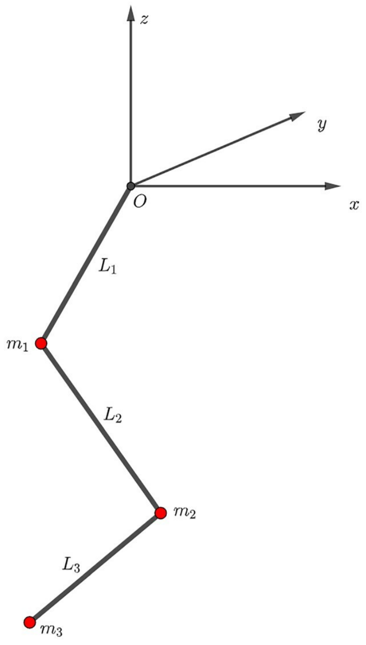

Figure 2.4: Threefold pendulum at a fixed time instant, with fixed point placed at the origin.

The Lagrangian we consider is a function from $\left(TS^2\right)^N$ to $\mathbb{R}$. 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,

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

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 $\omega_i^\top q_i = 0$ $\left(\text{i.e., } \omega_i \in T_{q_i}S^2\right)$ and pass to the coordinates $(q_1, \omega_1, q_2, \omega_2, ..., q_N, \omega_N) \in \left(TS^2\right)^N$ 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 $\left(TS^2\right)^N$

We characterize a transitive action for $\left(TS^2\right)^N$, 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

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

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

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

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

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

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

into other points of $TS^2_{|q|}$. Moreover, with standard arguments (see [43]), it is possible to prove that the orbit of a generic point $m = (q, \omega) \in \mathbb{R}^6$ with $\omega^\top q = 0$ 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

$$\psi : SE(3) \times TS^2 \to TS^2,$$
$$\psi\left((A,a),(q,\omega)\right) := \psi_{(A,a)}(q,\omega) = (Aq, A\omega + \hat{a}Aq) = (\bar{q},\bar{\omega}).$$

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

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

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

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

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

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

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

$$\psi : (SE(3))^N \times \left(TS^2\right)^N \to \left(TS^2\right)^N,$$
$$\psi\left((A_1, a_1, ..., A_N, a_n), (q_1, \omega_1, ..., q_N, \omega_N)\right) =$$
$$= (A_1 q_1, A_1\omega_1 + \hat{a}_1 A_1 q_1, ..., A_N q_N, A_N\omega_N + \hat{a}_N A_N q_N),$$

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 $m = (q_1, \omega_1, ..., q_N, \omega_N) \in \left(TS^2\right)^N$. 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 $F \in \mathfrak{X}\left(\left(TS^2\right)^N\right)$, 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 $F : \left(TS^2\right)^N \to \mathfrak{se}(3)^N$ such that

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

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 $\left(S^2\right)^N$ of [28]. The configuration manifold of the system is $\left(S^2\right)^N$, while the Lagrangian, expressed in terms of the variables $\left(q_1, \omega_1, ..., q_N, \omega_N\right) \in \left(TS^2\right)^N$, writes

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

where

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

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

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

we simplify the notation writing

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

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

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

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

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

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

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

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

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

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

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

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

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

where

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

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

$$0 \le \tilde{T}(q,\dot{q}) = \frac{1}{2}\sum_{i=1}^{N}\left(\sum_{j\ge i} m_j\right) L_i L_j \dot{q}_i^\top \dot{q}_j := \frac{1}{2}\dot{q}^\top M \dot{q}.$$

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

and $a_1,...,a_N : \left(TS^2\right)^N \to \mathbb{R}^3$ 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 $\left(TS^2\right)^N$ to $(\mathfrak{se}(3))^N$ such that

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

is the following one,

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

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 $\left(S^2\right)^N$ 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

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

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

2.5.2.1 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

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

$$R(q)\begin{bmatrix}\dot{\omega}_1\\ \dot{\omega}_2\end{bmatrix}=\begin{bmatrix}\left(-m_2L_1L_2|\omega_2|^2\hat{q}_2+(m_1+m_2)gL_1\hat{e}_3\right)q_1\\ \left(-m_2L_1L_2|\omega_1|^2\hat{q}_1+m_2gL_2\hat{e}_3\right)q_2\end{bmatrix}, \tag{2.5.1}$$

where
$$R(q)=\begin{bmatrix}(m_1+m_2)L_1^2I_3 & m_2L_1L_2\hat{q}_1^\top\hat{q}_2\\ m_2L_1L_2\hat{q}_2^\top\hat{q}_1 & m_2L_2^2I_3\end{bmatrix}.$$

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:

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

We can easily see that it is well defined since

$$R(q)\begin{bmatrix}\omega_1\\ \omega_2\end{bmatrix}=\begin{bmatrix}(m_1+m_2)L_1^2I_3 & m_2L_1L_2\hat{q}_1^\top\hat{q}_2\\ m_2L_1L_2\hat{q}_2^\top\hat{q}_1 & m_2L_2^2I_3\end{bmatrix}\begin{bmatrix}\hat{v}_1q_1\\ \hat{v}_2q_2\end{bmatrix}=\begin{bmatrix}\hat{r}_1q_1\\ \hat{r}_2q_2\end{bmatrix}\in\left(TS^2\right)^2$$

with
$$r_1(q,\omega):=(m_1+m_2)L_1^2v_1+m_2L_1L_2\hat{q}_2\hat{v}_2q_2,$$
$$r_2(q,\omega):=m_2L_1L_2\hat{q}_1\hat{v}_1q_1+m_2L_2^2v_2.$$

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

$$\dot{\omega}=R^{-1}(q)\begin{bmatrix}\left(-m_2L_1L_2|\omega_2|^2\hat{q}_2+(m_1+m_2)gL_1\hat{e}_3\right)q_1\\ \left(-m_2L_1L_2|\omega_1|^2\hat{q}_1+m_2gL_2\hat{e}_3\right)q_2\end{bmatrix}=R^{-1}(q)b=$$
$$=A^{-1}_{(q_1,q_2)}(b)=\begin{bmatrix}h_1\\ h_2\end{bmatrix}\in T_{q_1}S^2\times T_{q_2}S^2,$$

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 $h_i = h_i\left(q,\omega\right)$ and

$$\begin{bmatrix} h_1\left(q,\omega\right) \\ h_2\left(q,\omega\right) \end{bmatrix} = R^{-1}\left(q\right) \begin{bmatrix} \left(-m_2L_1L_2|\omega_2|^2\hat{q}_2 + \left(m_1+m_2\right)gL_1\hat{e}_3\right)q_1 \\ \left(-m_2L_1L_2|\omega_1|^2\hat{q}_1 + m_2gL_2\hat{e}_3\right)q_2 \end{bmatrix}.$$

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

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

through the function

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

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:

- $q_i(t)^\top q_i(t) = 1$, i.e., $q_i(t)\in S^2$, $i=1,2$,
- $q_i(t)^\top \omega_i(t) = 0$, 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

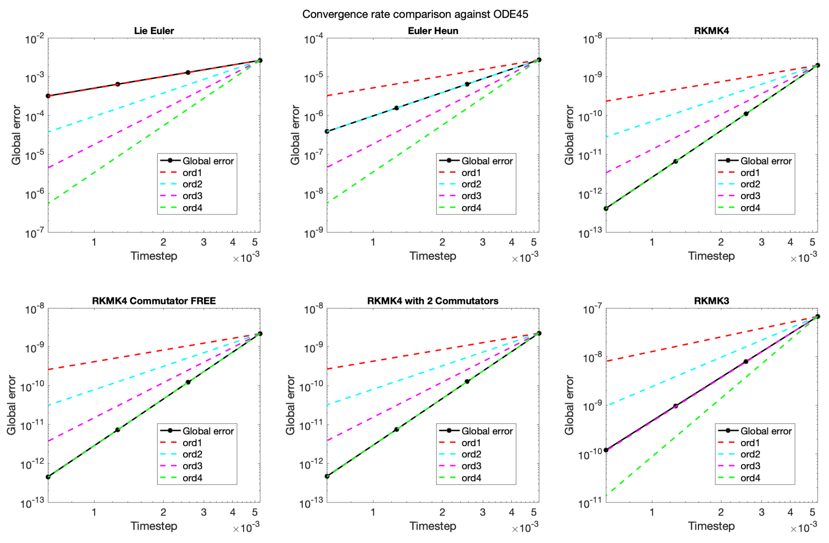

Figure 2.5: Convergence rate of the implemented Lie group integrators, based on global error considering as a reference solution the one of ODE45, with strict tolerance.

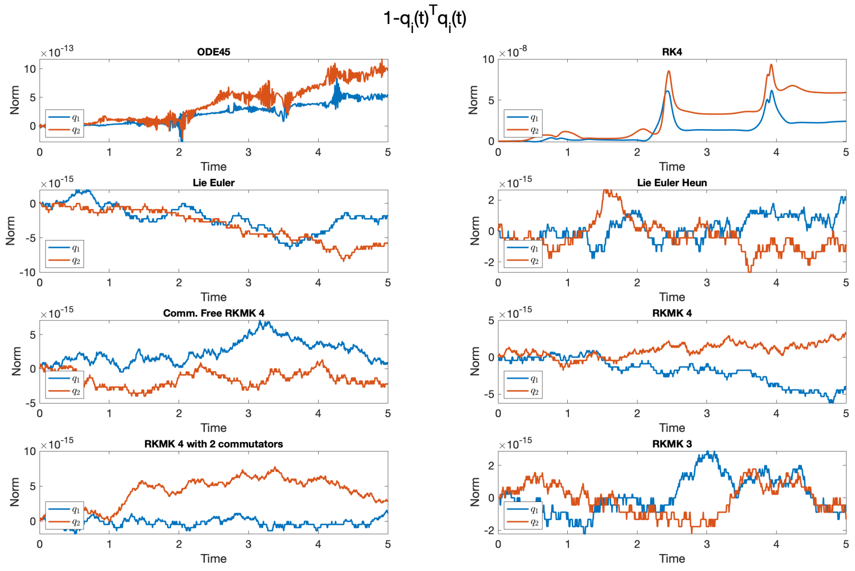

Figure 2.6: Visualization of the quantity $1 - q_i(t)^{\top} q_i(t)$, $i = 1,2$, for time $t \in [0,5]$. These plots focus on the preservation of the geometry of S^2.

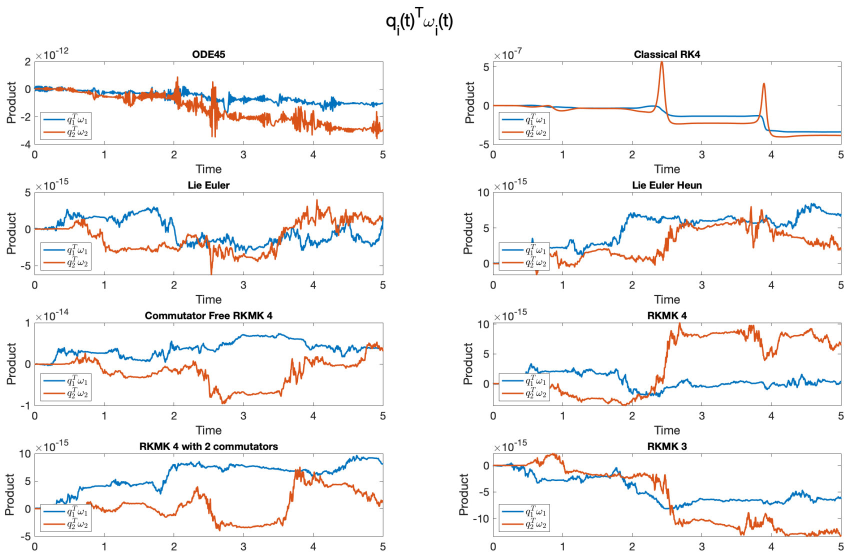

Figure 2.7: Visualization of the inner product $q_i(t)^\top \omega_i(t)$, $i = 1,2$, for $t \in [0,5]$. These plots focus on the preservation of the geometry of $T_{q_i(t)}S^2$.

we can move to the analysis of the time evolution on $TS^2 \times TS^2$. 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 $q_1(t)^\top \omega_1(t)$ and $q_2(t)^\top \omega_2(t)$ 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 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 $\mathbb{R}^{6N}$.

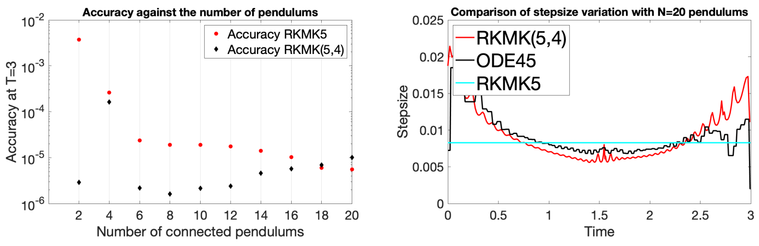

Figure 2.8: Comparison of accuracy at final time (on the left)and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 1$.

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 $\left(q_i, \omega_i\right) = \left(\sqrt{2}/2, 0, \sqrt{2}/2, 0, 1, 0\right), \forall i = 1, ..., N$, 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 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.

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 2.11 (right). In the latter, we plot two of the vector field components, i.e., the second components of the angular accelerations $\dot{\omega}_i(t)$, $i = 1, 2$. They show an

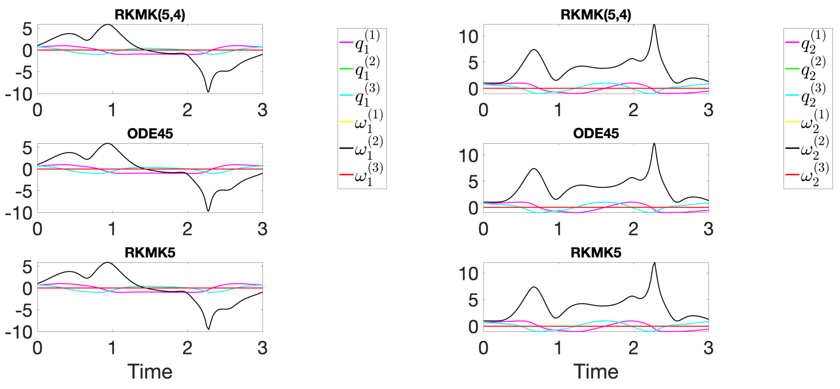

Figure 2.9: $\left(q_1(t), \omega_1(t)\right)$ (left), $\left(q_2(t), \omega_2(t)\right)$ (right). In these plots we represent the six components of the solution describing the dynamics of the first mass (on the left)and of the second mass (on the right), for the case $N = 2$. We compare the behaviour of the solution obtained with constant stepsize RKMK5, the variable stepsize RKMK(5,4) and ODE45.

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.

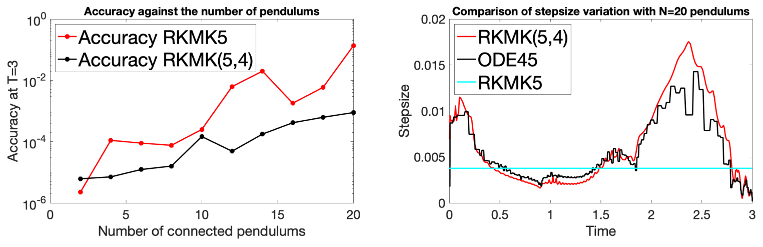

Figure 2.10: Comparison of accuracy at final time (on the left)and step adaptation for the case $N = 20$ (on the right), with all pendulums of length $L_i = 5/N$.

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

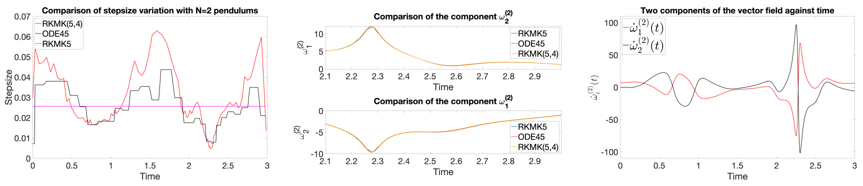

Figure 2.11: Step adaptation (left), Zoom at final times (middle), Values of $\dot{\omega}_i^{(2)}(t)$ (right). On the left, we compare the adaptation of the stepsize of RKMK(5,4)with the one of ODE45 and with the constant stepsize of RKMK5. In the center we plot the second component of the angular velocities $\omega_i^{(2)}$, $i=1,2$, 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.

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.

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, 29].

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 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, $(q_1,q_2)\in\left(S^2\right)^2$. The configuration manifold of the system is $Q=\mathbb{R}^3\times(SO(3))^2\times\left(S^2\right)^2$. In order to present the equations 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 $q_1,q_2\in S^2$,

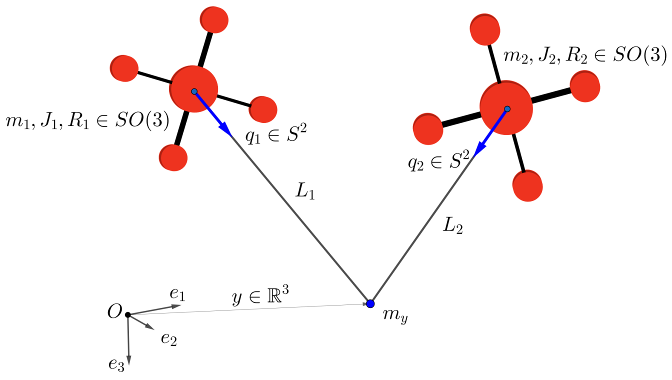

Figure 2.12: Two quadrotors connected to the mass point m_y via massless links of lengths L_i.

respectively, in terms of angular velocities, expressed with respect to the body-fixed frames. From these equations we define the trivialized Lagrangian

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

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

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

and

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

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

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

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

$$h(z)=\begin{bmatrix} h_1(z)\\ h_2(z)\\ h_3(z)\\ h_4(z)\\ h_5(z)\\ h_6(z)\end{bmatrix}=\begin{bmatrix} v\\ -\sum_{i=1}^{2} m_iL_i\|\omega_i\|^2 q_i+M_q g e_3+\sum_{i=1}^{2}u_i^{\parallel}\\ -\Omega_1\times J_1\Omega_1+M_1\\ -\Omega_2\times J_2\Omega_2+M_2\\ -\frac{1}{L_1}g\hat{q}_1e_3-\frac{1}{m_1L_1}q_1\times u_1^{\perp}\\ -\frac{1}{L_2}g\hat{q}_2e_3-\frac{1}{m_2L_2}q_2\times u_2^{\perp}\end{bmatrix},$$

where $M_q=m_yI_3+\sum_{i=1}^{2}m_iq_iq_i^{\top}$, 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., $u_i^{\parallel}=q_iq_i^Tu_i$, $u_i^{\perp}=\left(I-q_iq_i^T\right)u_i$. 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

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

2.6.1 Analysis via transitive group actions

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

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

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

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

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

$$\psi_g(P)=\left[y+a_1,\ v+a_2,\ B_1R_1,\ \Omega_1+b_1,\ B_2R_2,\ \Omega_2+b_2,\right.$$
$$\left.C_1q_1,\ C_1\omega_1+c_1\times C_1q_1,\ C_2q_2,\ C_2\omega_2+c_2\times C_2q_2\right].$$

The well-definiteness and transitivity of this action come from standard arguments, see for example [43]. The infinitesimal generator associated to

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

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

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

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

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

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

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

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

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

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

The above construction is completely independent of the control functions $\left\{u_i^{\parallel}, u_i^{\perp}, M_i\right\}_{i=1,2}$ 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 [29]. The control inputs $\left\{u_i^{\parallel}, u_i^{\perp}, M_i\right\}_{i=1,2}$ 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.

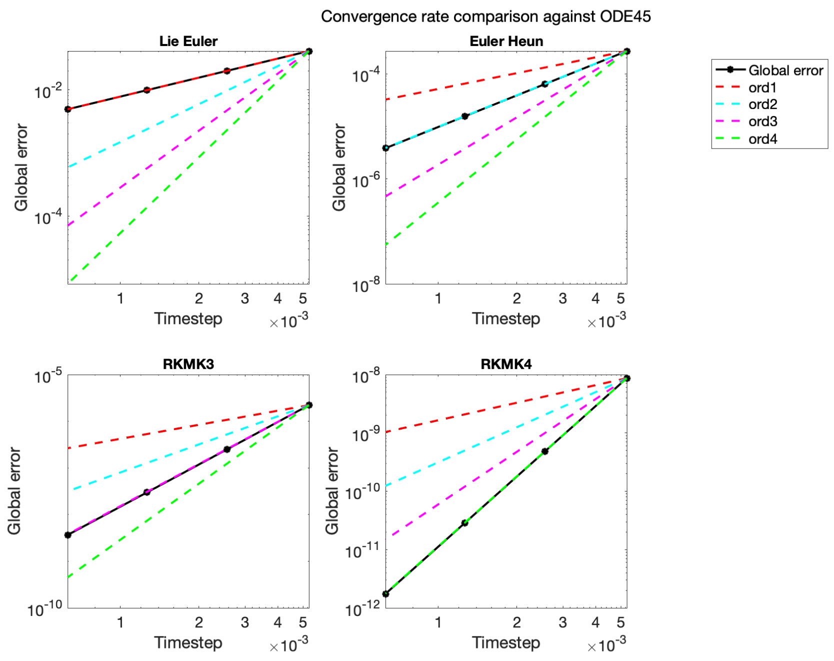

Figure 2.13: Convergence rate of the numerical schemes compared with ODE45

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.

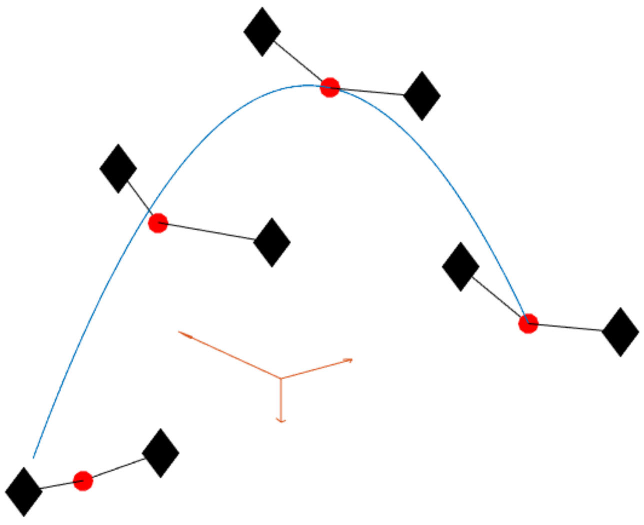

Figure 2.14: Snapshots at $0 \leq t \leq 5$.

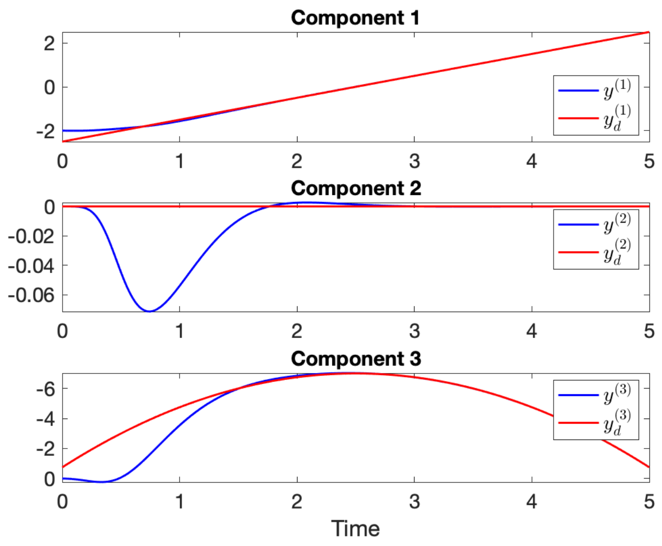

Figure 2.15: Components of the load position (in blue) and the desired trajectory (in red) as a function time.

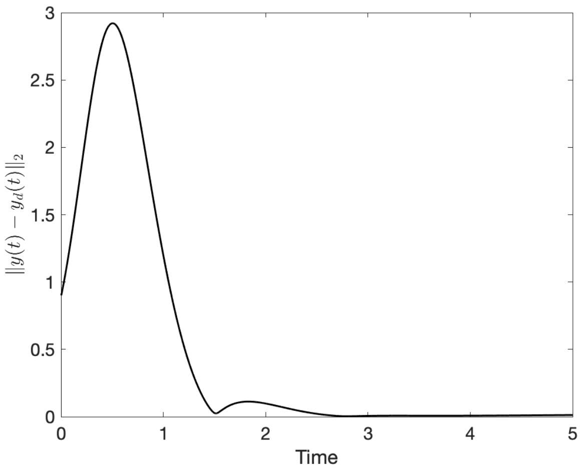

Figure 2.16: Deviation of the load position from the target trajectory.

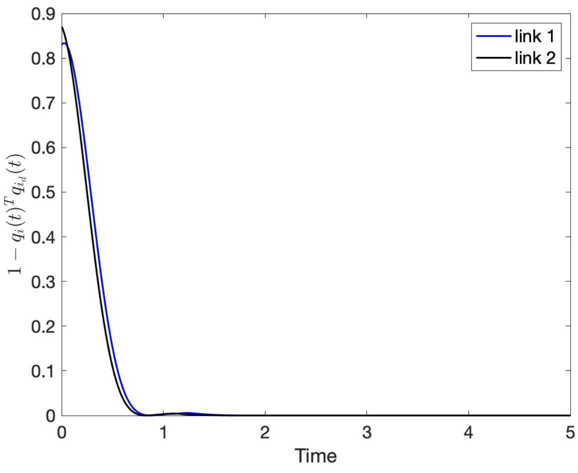

Figure 2.17: Direction error of the links.

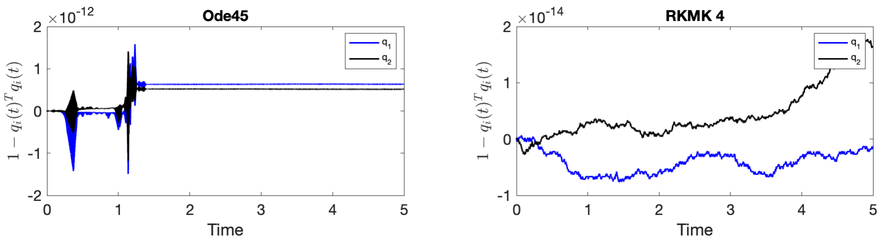

Figure 2.18: Preservation of the norms of $q_1, q_2 \in S^2$.

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 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 [41] 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.

Disclosure statement No potential conflict of interest was reported by the author(s).

Funding This work was supported by Marie Skłodowska-Curie [860124].

Bibliography

[1] M. Arnold and O. Brüls, *Convergence of the generalized-α scheme for constrained mechanical systems*, Multibody System Dynamics **18** (2007), no. 2, 185–202.

[2] M. Arnold, O. Brüls, and A. Cardona, *Error analysis of generalized-α Lie group time integration methods for constrained mechanical systems*, Numerische Mathematik **129** (2015), no. 1, 149–179.

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

[4] O. Bruls and A. Cardona, *On the Use of Lie Group Time Integrators in Multibody dynamics*, Journal of Computational Nonlinear Dynamics **5** (2010), no. 3, 031002.

[5] F. Casas and B. Owren, *Cost Efficient Lie Group Integrators in the RKMK Class*, BIT Numerical Mathematics **43** (2003), no. 4, 723–742.

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

[7] E. Celledoni, H. Marthinsen, and B. Owren, *An introduction to Lie group integrators—basics, new developments and applications*, Journal of Computational Physics **257** (2014), 1040–1061.

[8] E. Celledoni and B. Owren, *Lie group methods for rigid body dynamics and time integration on manifolds*, Computer Methods in Applied Mechanics and Engineering **192** (2003), no. 3-4, 421–438.

[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 Humanoid Robotics, Cancun, Mexico, 15–17 November 2016 (2016), 826–833.

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

[11] P. E. Crouch and R. Grossman, *Numerical integration of ordinary differential equations on manifolds*, Journal of Nonlinear Science **3** (1993), 1–33.

[12] C. Curry and B. Owren, *Variable step size commutator free Lie group integrators*, Numerical Algorithms **82** (2019), no. 4, 1359–1376.

[13] F. Diele, L. Lopez, and R. Peluso, *The Cayley transform in the numerical solution of unitary differential systems*, Advances in Computational Mathematics **8** (1998), no. 4, 317–334.

[14] J. R. Dormand and P. J. Prince, *A family of embedded Runge-Kutta formulae*, Journal of Computational and Applied Mathematics **6** (1980), no. 1, 19–26.

[15] K. Engø and S. Faltinsen, *Numerical integration of Lie–Poisson systems while preserving coadjoint orbits and energy*, SIAM Journal on Numerical Analysis **39** (2001), no. 1, 128–145.

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

[17] V. Guillemin and S. Sternberg, *The moment map and collective motion*, Annals of Physics **127** (1980), 220–253.

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

[19] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations I, Nonstiff problems*, Second revised ed., Springer-Verlag, 1993.

[20] J. Hall and M. Leok, *Lie group spectral variational integrators*, Foundations of Computational Mathematics **17** (2017), no. 1, 199–257.

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

[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** (1977), no. 3, 283–292.

[23] D. Holm, *Geometric mechanics: Part II: Rotating, translating and rolling*, World Scientific Publishing Company, 2008.

[24] D. Holm, J. Marsden, and T. Ratiu, *The Euler–Poincaré equations and semidirect products with applications to continuum theories*, Advances in Mathematics **137** (1998), 1–81.

[25] S. Holzinger and J. Gerstmayr, *Time integration of rigid bodies modelled with three rotation parameters*, Multibody System Dynamics (2021), 1–34.

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

[27] T. Lee, M. Leok, and N. H. McClamroch, *Lie group variational integrators for the full body problem*, Computer Methods in Applied Mechanics and Engineering **196** (2007), no. 29-30, 2907–2924.

[28] ______, *Global formulations of Lagrangian and Hamiltonian dynamics on manifolds*, Interaction of Mechanics and Mathematics, Springer, Cham, 2018, A geometric approach to modeling and analysis.

[29] T. Lee, K. Sreenath, and V. Kumar, *Geometric control of cooperating multiple quadrotor UAVs with a suspended payload*, 52nd IEEE Conference on Decision and Control (2013), 5510–5515.

[30] T. Leitz and S. Leyendecker, *Galerkin Lie-group variational integrators based on unit quaternion interpolation*, Computer Methods in Applied Mechanics and Engineering **338** (2018), 333–361.

[31] N. E. Leonard and J. E. Marsden, *Stability and drift of underwater vehicle dynamics: Mechanical systems with rigid motion symmetry*, Physica D **105** (1997), no. 1–3, 130–162.

[32] D. Lewis and J. C. Simo, *Conserving algorithms for the dynamics of Hamiltonian systems of Lie groups*, Journal of Nonlinear Science **4** (1994), 253–299.

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

[34] J. E. Marsden, T. Ratiu, and A. Weinstein, *Semi-direct products and reduction in mechanics*, Journal of Geometry and Physics **281** (1884), 147–77.

[35] J. E. Marsden and T. S. Ratiu, *Introduction to mechanics and symmetry*, Springer-Verlag, 1994.

[36] ———, *Introduction to mechanics and symmetry*, second ed., Texts in Applied Mathematics, no. 17, Springer-Verlag, 1999.

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

[38] J. Moser and A. P. Veselov, *Discrete versions of some classical integrable systems and factorization of matrix polynomials*, Communications in Mathematical Physics **139** (1991), no. 2, 217–243.

[39] A. Müller, *Coordinate mappings for rigid body motions*, ASME Journal of Computational and Nonlinear Dynamics **12** (2017), no. 2, 021010.

[40] H. Munthe-Kaas, *Runge–Kutta methods on Lie groups*, BIT Numerical Mathematics **38** (1998), no. 1, 92–111.

[41] ———, *High order Runge–Kutta methods on manifolds*, Applied Numerical Mathematics **29** (1999), no. 1, 115–127.

[42] H. Munthe-Kaas and B. Owren, *Computations in a free Lie algebra*, Philosophical Transactions of the Royal Society A **357** (1999), 957–981.

[43] P. J. Olver, *Applications of Lie groups to differential equations*, vol. 107, Springer Science & Business Media, 2000.

[44] B. Owren, *Order conditions for commutator-free Lie group methods*, Journal of Physics **39** (2006), no. 19, 5585–5599.

[45] ———, *Lie group integrators*, Discrete mechanics, geometric integration and Lie-Butcher series, Springer Proc. Math. Stat., vol. 267, Springer, Cham, 2018, pp. 29–69.

[46] B. Owren and A. Marthinsen, *Integration methods based on canonical coordinates of the second kind*, Numer. Math. **87** (2001), no. 4, 763–790.

[47] J. Park and W. Chung, *Geometric integration on Euclidean group with application to articulated multibody systems*, Journal of Computational and Applied Mathematics **21** (2005), no. 5, 850–863.

[48] T. Ratiu, *Euler-Poisson equations on Lie algebras and the N-dimensional heavy rigid body*, Proceedings of the National Academy of Sciences (1981), 1327–1328.

[49] A. Saccon, *Midpoint rule for variational integrators on Lie groups*, International Journal for Numerical Methods in Engineering **78** (2009), no. 11, 1345–1364.

[50] J. C. Simo and L. Vu-Quoc, *On the dynamics of finite-strain rods undergoing large motions – A geometrically exact approach*, Computer Methods in Applied Mechanics and Engineering **66** (1988), 125–161.

[51] A. P. Veselov, *Integrable systems with discrete time, and difference operators*, Funktsional. Anal. i Prilozhen. **22** (1988), no. 2, 1–13, 96.

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

[53] F. W. Warner, *Foundations of differentiable manifolds and Lie groups*, GTM 94, Springer-Verlag, 1983.

[54] V. Wieloch and M. Arnold, *BDF integrators for constrained mechanical systems on Lie groups*, Journal of Computational and Applied Mathematics **387** (2019), 112517.

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

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

In: M. Ehrhardt, M. Günther (eds) Progress in Industrial Mathematics at ECMI 2021. ECMI 2021. Mathematics in Industry(), Vol 39. Springer, Cham., 2022, pp. 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 Section 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 $F|_m = \psi_* (f(m))\big|_m$, 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

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

where $\sigma_1 \approx \sigma\left(h_n\right) \in \mathfrak{g}$ 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, $e_{n+1} = Ch_{n+1}^{\tilde{p}+1} + \mathcal{O}\left(h^{\tilde{p}+2}\right)$. Aiming at $e_{n+1} \approx \text{tol}$ in every step, one may use a formula of the type

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

where θ is typically chosen between 0.8 and 0.9. If $e_n > \text{tol}$, 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 $\Lambda\left(g, q_1\right) = q_2$. Assume that for each $q \in Q$, the map $\Lambda_q : G \to Q$ defined as $\Lambda_q\left(g\right) := \Lambda\left(g, q\right)$, 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 $\bar{\xi}\left(q\right) := \Lambda_*\left(\xi\right)\left(q\right) \in T_qQ$ the differential at the identity element $e \in G$ of Λ_q, evaluated at $\xi \in \mathfrak{g}$. We then introduce $\Lambda_g : Q \to Q,\ q \mapsto \Lambda\left(g, q\right)$ 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:

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

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

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

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

$$\Lambda : SO(3) \times S^2 \to S^2,\ \left(R, q\right) \mapsto Rq,$$
$$T_qS^2 \ni \Lambda_*\left(\xi\right)\left(q\right) = \bar{\xi}\left(q\right) = \xi \times q, \quad T_q\Lambda_R\left(v\right) = Rv \in T_{Rq}S^2.$$

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]),

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

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 $G \ni g_i = \left(g_i^1, ..., g_i^n\right)$, $i = 1, 2$, we can define their product $g_1 \cdot g_2 := \left(g_1^1 \odot_1 g_2^1, ..., g_1^n \odot_n g_2^n\right) \in G$. 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

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

then

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

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

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

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

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 $\left(TS^2\right)^N$.

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

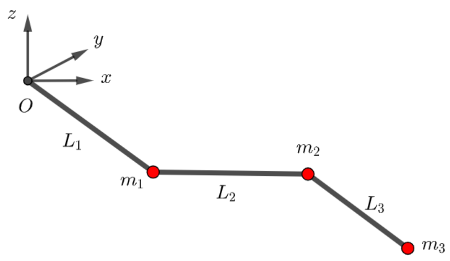

Figure 3.1: Chain of 3 connected pendulums at a fixed time instant.

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 [9], we express the Euler–Lagrange equations for our system in terms of the configuration variables $(q_1, \dots, q_N) \in \left(S^2\right)^N \subset \mathbb{R}^{3N}$, 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

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

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

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

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

and

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

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

$\mathfrak{se}(3)^N$ such that

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

where ψ is defined as in Section 3.3.2.

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

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

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

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

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

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

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 $(q_i, \omega_i) = (1,0,0,0,0,0)$, $\forall i = 1,\ldots,N$. As we can see in Figure 3.2 (left), 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.

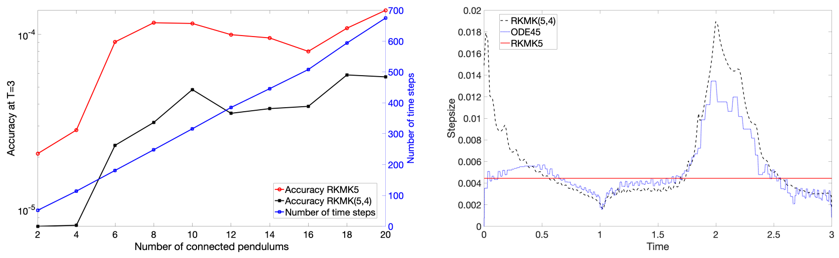

Figure 3.2: Comparisons of variable versus constant stepsize for the N-fold 3D pendulum. Accuracy against the number of pendulums (left), Comparison of step sizes with 20 pendulums (right).

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

Acknowledgements 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** (1972), no. 2, 550–551.

[2] E. Celledoni, E. Çokaj, A. Leone, D. Murari, and B. Owren, *Lie group integrators for mechanical systems*, International Journal of Computer Mathematics **99** (2022), no. 1, 58–88.

[3] E. Celledoni, H. Marthinsen, and B. Owren, *An introduction to Lie group integrators–basics, new developments and applications*, Journal of Computational Physics **257** (2014), 1040–1061.

[4] J. R. Dormand and P. J. Prince, *A family of embedded Runge-Kutta formulae*, Journal of Computational and Applied Mathematics **6** (1980), no. 1, 19–26.

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

[6] B. C. Hall, *Lie groups, Lie algebras, and representations: An elementary introduction*, Graduate Texts in Mathematics, Springer, 2015.

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

[8] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna, *Lie-group methods*, Acta numerica **9** (2000), 215–365.

[9] T. Lee, M. Leok, and N. H. McClamroch, *Lagrangian and Hamiltonian dynamics on manifolds*, Global Formulations of Lagrangian and Hamiltonian Dynamics on Manifolds: A Geometric Approach to Modeling and Analysis (2018), 347–398.

B-stability of numerical integrators on Riemannian manifolds

Martin Arnold, Elena Celledoni, Ergys Çokaj,
Brynjulf Owren, and Denise Tumiotto

Journal of Computational Dynamics Vol. 11, No. 1, 2024, pp. 92-107.

B-stability of numerical integrators on Riemannian manifolds

Abstract. We propose a generalization of nonlinear stability of numerical one-step integrators to Riemannian manifolds in the spirit of Butcher's notion of B-stability. Taking inspiration from Simpson-Porco and Bullo, we introduce non-expansive systems on such manifolds and define B-stability of integrators. In this first exposition, we provide concrete results for a geodesic version of the Implicit Euler (GIE) scheme. We prove that the GIE method is B-stable on Riemannian manifolds with non-positive sectional curvature. We show through numerical examples that the GIE method is expansive when applied to a certain non-expansive vector field on the 2-sphere, and that the GIE method does not necessarily possess a unique solution for large enough step sizes. Finally, we derive a new improved global error estimate for general Lie group integrators.

4.1 Introduction

Stability is a fundamental property of numerical methods for stiff nonlinear ordinary differential equations. It is important for controlling the growth of error in the numerical approximation and is used in combination with local error estimates to obtain bounds for the global error. Stability bounds can also in some situations be used to ensure the existence and uniqueness of a solution to the algebraic equations arising from implicit integrators. In the literature, one can find a large variety of stability definitions for numerical integrators with various different aims. Some of them apply to linear test equations, others are of a more general nature and apply to nonlinear problems with certain prescribed properties. Most of the stability definitions found in the literature are developed for problems modeled on linear spaces. In particular, there is a well-established non-linear stability theory, where an inner product norm is used to measure the distance between two solutions and the corresponding numerical approximations. Pioneering contributions to this theory were made by Dahlquist and Butcher in the mid-1970s [6,17], in the wake of the legendary numerical analysis conference in Dundee, 1975. The notions of G-stability for multi-step methods [17] and B-stability of Runge–Kutta methods [6] were developed. The overall idea of B-stability is that whenever the norm of the difference between two solutions of the ODE is monotonically non-increasing, the numerical method should exhibit a similar behavior, that is, the difference in norm between the two corresponding numerical solutions should not increase over a time step. Much is known about B-stable Runge–Kutta methods, and there is even an algebraic condition on the coefficients (A, b) of a method that ensures its B-stability. A key ingredient is the one-sided Lipschitz condition, also called a monotonicity condition, on the ODE vector field. We refer the reader to the excellent monographs [20, 24] for a detailed treatment of the various definitions of stability and B-stability in particular.

We remark that whether a particular ODE system is non-expansive depends on the choice of inner product norm, but the notion of a B-stable Runge–Kutta method does not, see [24, p. 182]. In this paper, we shall be concerned with *unconditional stability*, meaning that step sizes $h \in (0, \infty)$ are allowed. This excludes all explicit integrators, and it makes it necessary to assume that both the flow of the ODE vector field and the numerical method map are well defined for all positive t. Dahlquist and Jeltsch [18] introduced *generalized disks of contractivity* in order to consider also the case in which limitations on the ODE vector field and the step size are imposed.

We shall here consider systems of ODEs whose solutions evolve on a smooth manifold. We are primarily interested in numerical integrators which are intrinsic, that are not developed for a particular choice of local coordinates, or based on a specific embedding of the manifold into an ambient space. There are several such numerical methods available in the literature.

Crouch and Grossman [15] proposed to build integrators by composing flows of so-called frozen vector fields, and these methods were later extended to a more general format in [10] called *Commutator-free Lie group methods*. Munthe–Kaas introduced numerical integrators for homogeneous spaces [41] by equipping the manifold with a left transitive Lie group action which was used together with the exponential map to transform the ODE vector field locally to a vector field on the underlying Lie algebra. Its flow is approximated by any classical Runge–Kutta method, and the result is mapped back to the manifold by composing the group action with the exponential map.

In computational mechanics there were early contributions to numerical integration on particular manifolds, such as the rotation group $SO(3)$ and the special Euclidean group $SE(3)$. A landmark paper in the design of conservative methods for Hamiltonian systems on Lie groups is the one by Lewis and Simo [37]. For rod dynamics, an important paper was that of Simo and Vu-Quoc [46] who developed a geometrically exact formulation for rods undergoing large motions, and for the time stepping they devised a version of the Newmark methods applicable to Lie groups. These methods can be generalized to the so-called α-methods [27] in a Lie group setting, see [2, 3]. Parametrization of the manifold in question, such as the rotation group, plays a significant role in computational mechanics, for efficiency, accuracy, and storage requirements. When using (minimal) local coordinates for global simulation, one inevitably runs into problems with singularities, these issues have been studied and amended by several authors, e.g. [28, 48]. Hamiltonian systems are often formulated on cotangent bundles, in which case symplectic integrators can be derived through the discretization of a variational problem, this approach is sometimes named discrete mechanics. The pioneering work by Marsden and West [38] developed this theory for Euclidean spaces, and it has later been generalized to Lie groups in a number of papers [5, 11, 21, 25, 26, 34, 36].

Finally, on a Riemannian manifold, it is natural to base the numerical schemes primarily on the Riemannian exponential map. Leimkuhler and Patrick [35] derived a symplectic integrator for Riemannian manifolds, and in [12] the authors suggest using Riemannian normal coordinates to define a retraction map.

For an in-depth account of Lie group methods, we refer to [7, 11, 14, 30, 42] and references therein.

In this paper we shall make the first attempt to generalize B-stability to Riemannian manifolds, replacing the inner product norm with the Riemannian distance function. We take inspiration from the work of Simpson-Porco and Bullo [47] who considered contraction properties of a continuous system. In Section 4.2 we define what we mean by a non-expansive system on a Riemannian manifold, and we state the definition of B-stability of a general numerical method in this setting. Then, in Section 4.3 we first present two examples of numerical methods: the geodesic versions of the implicit Euler method (GIE) and the implicit midpoint rule (GIMP). Then we prove a B-stability result for the GIE method in the case that the manifold has non-positive sectional curvature. We also provide numerical experiments for a particular vector field on the two-sphere, S^2, showing that neither the GIE nor the GIMP method is B-stable on this manifold which has positive sectional curvature. We briefly discuss also for this example a non-uniqueness issue with the GIE method which is different from what is known from the Euclidean setting. Finally, in Section 4.4 we present a bound for the global error of numerical methods, based on the monotonicity condition.

4.2 Non-expansive systems

We begin by briefly introducing some notation and terminology, mostly adhering to the monograph by Lee [33]. A Riemannian manifold is a pair (M, g), where M is a smooth manifold and g is a smoothly varying inner product defined on each tangent space $T_pM,\ p \in M$. We will use interchangeably the notations $g(\cdot,\cdot)$ and $\langle\cdot,\cdot\rangle$. Associated to (M, g) is the Levi-Civita connection, the unique affine connection ∇, which for any three vector fields X, Y, Z on M satisfies $X\langle Y, Z\rangle = \langle \nabla_X Y, Z\rangle + \langle Y, \nabla_X Z\rangle$ and $[X, Y] = \nabla_X Y - \nabla_Y X$. The connection also defines the covariant derivative of vector fields along curves, we use the notation $D_t V(t)$ to denote the covariant derivative of $V(t)$ along $\gamma(t)$, see [33, Theorem 4.21]. A curve $\gamma : [a, b] \to M$ is geodesic if it satisfies the equation $D_t\dot{\gamma}(t) = 0$ along $\gamma(t)$. A geodesic that connects two points p

and q is called a geodesic segment. If this second order differential equation, together with initial data $\gamma(0) = p \in M$, $\dot{\gamma}(0) = v_p \in T_pM$ yields a solution $\gamma(t)$, $t \in [0, t^*]$, we use the notation $\gamma(t) = \exp_p\left(tv_p\right)$, thus $\exp_p : T_pM \to M$. A similar notation is used for the t-flow, $\exp(tX)$, of a vector field X on M, it is the diffeomorphism on M, $p \mapsto y(t)$ where $\dot{y} = X|_y$, $y(0) = p$, and its domain of definition may be t-dependent. A numerical method on M is a map $\phi_{t,X} : M \to M$ that approximates the flow map $\exp(tX)$. A set $\mathcal{U} \subseteq M$ is geodesically convex if, for each $p, q \in \mathcal{U}$, there is a unique minimizing geodesic segment from p to q contained entirely in $\mathcal{U}$. A vector field X is forward complete on $\mathcal{U}$ if for every $p \in \mathcal{U}$, $\exp(tX)p$ is defined for all $t \geq 0$. If for every $(t, p) \in [0, \infty) \times \mathcal{U}$ it holds that $\exp(tX)p \in \mathcal{U}$, we say that $\mathcal{U}$ is forward X-invariant. Similarly, for a mapping ρ the set $\mathcal{U}$ is ρ-invariant if $\rho(y) \in \mathcal{U}$ for any $y \in \mathcal{U}$. We denote the length of a curve $\gamma : [a, b] \to M$ as $\ell(\gamma) = \int_a^b \|\dot{\gamma}(t)\| \, dt$, where $\|v\| := \langle v, v\rangle^{\frac{1}{2}}$ is the g-norm. The metric induces a distance function between pairs of points $p, q \in M$, $d(p, q) = \inf_{\gamma_{p\to q}} \ell\left(\gamma_{p\to q}\right)$, where $\gamma_{p\to q}$ is any continuous curve connecting p and q. The following definition replaces the one-sided Lipschitz condition on a Riemannian manifold.

Definition 4.1. Let $(M, \langle\cdot,\cdot\rangle)$ be a Riemannian manifold and let $\mathcal{U} \subset M$. We say that the vector field X satisfies a monotonicity condition on the set $\mathcal{U}$ with constant $\nu \in \mathbb{R}$ if for every $x \in \mathcal{U}$ and $v_x \in T_xM$, it holds that

$$\left\langle \nabla_{v_x} X, v_x \right\rangle \leq \nu \left\| v_x \right\|^2. \tag{4.2.1}$$

Consider for every $x \in \mathcal{U}$, the linear operator $\nabla X|_x : v_x \mapsto \nabla_{v_x} X$ on T_xM. The constant ν can be chosen as

$$\nu = \sup_{x \in \mathcal{U}} \mu_g\left(\nabla X|_x\right), \tag{4.2.2}$$

where μ_g is the logarithmic g-norm of $\nabla X|_x$. For a linear operator $A : T_xM \to T_xM$, its logarithmic g-norm is defined as [20]

$$\mu_g(A) = \sup_{0 \neq v \in T_xM} \frac{g(Av, v)}{g(v, v)}.$$

In local coordinates $\mathbf{x} = (x_1, \ldots, x_m)$ on M, we write the vector field as $X = X^i(\mathbf{x})\partial_i$ and the metric tensor g is represented by the matrix $\mathbf{g}(\mathbf{x})$ with elements $\mathbf{g}_{ij} = g\left(\partial_i, \partial_j\right)$. The operator ∇X has the matrix representation $\mathcal{A}(X)$ where $\mathcal{A}(X)_i^k = \partial_i X^k + \Gamma_{ij}^k X^j$ and where Γ_{ij}^k are the Christoffel symbols of the connection. We can now formulate the logarithmic g-norm of ∇X pointwise as

$$\mu_g(\nabla X) = \max \lambda \left[\mathbf{g}^{1/2}(\mathbf{x})\mathcal{A}(X)\mathbf{g}^{-1/2}(\mathbf{x}) + \mathbf{g}^{-1/2}(\mathbf{x})\mathcal{A}(X)^\top \mathbf{g}^{1/2}(\mathbf{x})\right],$$

i.e., the largest eigenvalue of the matrix in square brackets, see also [19].

Theorem 4.1. *Let (M,g) be a Riemannian manifold, $\mathcal{U} \subset M$ a geodesically convex set, and let X be a vector field on M satisfying the monotonicity condition* (4.2.1) *on $\mathcal{U}$ with a constant $\nu \in \mathbb{R}$. Suppose that for any $x_0, y_0 \in \mathcal{U}$, there is a $t^* > 0$ such that $\exp(tX)\,x_0$ and $\exp(tX)\,y_0$ exist and are contained in $\mathcal{U}$ for every $t \in [0, t^*]$. Then, it holds that*

$$d\Big(\exp(tX)\,x_0, \exp(tX)\,y_0\Big) \le d(x_0, y_0)\,\mathrm{e}^{\nu t} \quad \textit{for every } t \in [0, t^*]. \tag{4.2.3}$$

Remark 4. *The condition that the set $\mathcal{U}$ is geodesically convex can be weakened by introducing the notion of a K-reachable set as in [47].*

Proof. The construction for the proof is illustrated in Figure 4.1. Since $\mathcal{U}$ is geodesically convex, there is a unique minimizing geodesic $\gamma(s) \in \mathcal{U}$ connecting $x_0, y_0 \in \mathcal{U}$, with $\gamma(0) = x_0$ and $\gamma(1) = y_0$. We will be using the notation $\Gamma(s,t) := \exp(tX)\,\gamma(s)$, as in [33, Chapter 6], and $\Gamma(s,t)$ is contained in $\mathcal{U}$. For a fixed $t \in [0, t^*]$, consider the length $\ell(t)$ of the curve $s \mapsto \Gamma(s,t)$, $s \in [0,1]$, that is

$$\ell(t) = \int_0^1 \Big\langle \partial_s \Gamma(s,t), \partial_s \Gamma(s,t) \Big\rangle^{\frac{1}{2}} ds, \tag{4.2.4}$$

and we have $d(x_0, y_0) = \ell(0)$. Let

$$S(s,t) := \partial_s \Gamma(s,t), \quad T(s,t) := \partial_t \Gamma(s,t). \tag{4.2.5}$$

We will use that

$$D_t S(s,t) = D_s T(s,t) = D_s X\big|_{\Gamma(s,t)},$$

following from the symmetry lemma [33, Lemma 6.2]. Differentiating with respect to t, using the chain rule and the properties of the Levi-Civita connection, we have

$$\begin{aligned}
\frac{\mathrm{d}\ell(t)}{\mathrm{d}t} &= \int_0^1 \frac{\partial_t \Big\langle S(s,t), S(s,t) \Big\rangle}{2\Big\| S(s,t) \Big\|} ds = \int_0^1 \frac{\Big\langle D_t S(s,t), S(s,t) \Big\rangle}{\Big\| S(s,t) \Big\|} ds \\
&= \int_0^1 \frac{\Big\langle D_s T(s,t), S(s,t) \Big\rangle}{\Big\| S(s,t) \Big\|} ds = \int_0^1 \frac{\Big\langle D_s X(\Gamma(s,t)), S(s,t) \Big\rangle}{\Big\| S(s,t) \Big\|} ds \\
&\le \int_0^1 \frac{\nu \Big\langle S(s,t), S(s,t) \Big\rangle}{\Big\| S(s,t) \Big\|} ds = \nu\,\ell(t),
\end{aligned}$$

where the last inequality follows from the assumption that X satisfies the monotonicity condition (4.2.1). By Gronwall's lemma, we obtain the inequality

$$\ell(t) \le \ell(0)\,\mathrm{e}^{\nu t}, \qquad \text{for each } t \in [0, t^*],$$

and conclude that

$$d\left(\exp\left(tX\right)x_0, \exp\left(tX\right)y_0\right) \leq \ell\left(t\right) \leq \ell\left(0\right)\mathrm{e}^{\nu t} = d\left(x_0, y_0\right)\mathrm{e}^{\nu t}.$$

□

Remark 5. *Choosing* $\nu = \sup\left\{\left\|\nabla X|_p\right\| : p \in \Gamma\left(\left[0, t^*\right] \times \left[0, 1\right]\right)\right\}$ *leads to a bound similar to the one in Theorem 1.2 by Kunzinger et al. in [32].*

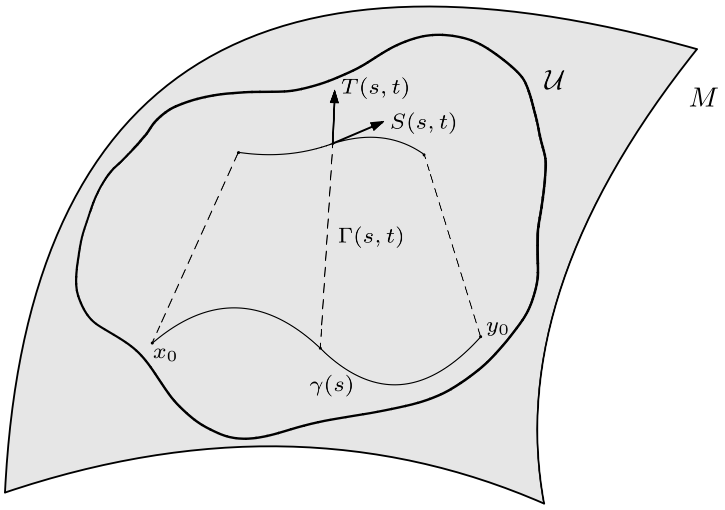

Figure 4.1: Construction for the proofs of Theorems 4.1 and 4.2.

The next definition is inspired by the definition of contracting systems by Simpson-Porco and Bullo in [47].

Definition 4.2 (Non-expansive system)**.** Let $\left(M, g\right)$ be a Riemannian manifold. Let $\mathcal{U} \subseteq M$ be an open, geodesically convex set and $X \in \mathfrak{X}(M)$. If

(i) X is forward complete on $\mathcal{U}$,

(ii) $\mathcal{U}$ is forward X-invariant,

(iii) X satisfies the monotonicity condition (4.2.1) on $\mathcal{U}$ with $\nu \leq 0$,

the quadruple $\left(\mathcal{U}, X, g, \nu\right)$ is called a non-expansive system.

We are now ready to give the definition of a B-stable numerical method on Riemannian manifolds.

Definition 4.3 (B-stability)**.** Let (M,g) be a Riemannian manifold and let $\phi_{h,X}$ be a numerical method on M. Suppose that for any non-expansive system $(\mathcal{U},X,g,\nu)$ on M, it holds that

(i) $\phi_{h,X}$ is forward complete on $\mathcal{U}$, i.e., $\phi_{h,X}$ is well defined for all $h>0$, and

(ii) $\mathcal{U}$ is forward $\phi_{h,X}$-invariant for all $h>0$.

If

$$d\left(\phi_{h,X}(x_0),\phi_{h,X}(y_0)\right)\le d(x_0,y_0),\quad x_0,y_0\in\mathcal{U},h>0,$$

then $\phi_{h,X}$ is called B-stable.

4.3 Numerical integrators on manifolds and B-stability

Geodesic Explicit Euler (GEE) method The simplest numerical method defined on a Riemannian manifold is the Geodesic Explicit Euler method

$$y_{n+1}=\exp_{y_n}\left(h\,X|_{y_n}\right),\tag{4.3.1}$$

that can not be unconditionally stable, but will be used for comparison in the numerical experiments in Example 3.2.

Geodesic Implicit Euler (GIE) method We consider the following definition of the Implicit Euler method in a Riemannian manifold

$$y_n=\exp_{y_{n+1}}\left(-h\,X|_{y_{n+1}}\right).\tag{4.3.2}$$

This reduces to the classical implicit Euler method when the manifold is the Euclidean space.

Geodesic Implicit Midpoint (GIMP) method Similarly, we consider the implicit midpoint rule on a Riemannian manifold:

$$\begin{aligned}y_n&=\exp_{\bar y}\left(-\frac{1}{2}h\,X|_{\bar y}\right),\\ y_{n+1}&=\exp_{\bar y}\left(\frac{1}{2}h\,X|_{\bar y}\right).\end{aligned}\tag{4.3.3}$$

This method can be found in Zanna et al. [50] for the case of Lie group integrators. It is a symmetric method, but it is not generally symplectic. In [39] a symplectic method was found for products of 2-spheres, $\left(S^2\right)^d$, that happens to be a time reparametrization of (4.3.3). It is called the *spherical midpoint method* (SPHMP). Applied to a single copy of S^2 it reads in Cartesian coordinates

$$y_{n+1}=y_n+h\,X|_{\bar y},\quad \bar y=\frac{y_n+y_{n+1}}{\|y_n+y_{n+1}\|}.\tag{4.3.4}$$

4.3.1 The case with non-positive sectional curvature

In the next theorem, we prove the B-stability of the GIE method on Hadamard manifolds, i.e., manifolds with non-positive sectional curvature.

Theorem 4.2 (B-stability of the GIE method)**.** *Let (M,g) be a Riemannian manifold with non-positive sectional curvature. Then, the GIE method* (4.3.2) *is B-stable.*

Proof. Let $(\mathcal{U},X,g,0)$ be a non-expansive system of ODEs, and consider $\phi_{h,X}$ with step size $h>0$. Let $\gamma_0(s), s\in[0,1]$ be a curve in $\mathcal{U}$ such that $\gamma_0(0)=x_0\in\mathcal{U}$ and $\gamma_0(1)=y_0\in\mathcal{U}$, and set $\gamma_1(s)=\phi_{h,X}(\gamma_0(s))$. By assumption $\gamma_1(s)$ is well defined and contained in $\mathcal{U}$. Consider the one-parameter family of curves

$$\Gamma(s,t):=\exp_{\gamma_1(s)}\left(-th\, X|_{\gamma_1(s)}\right).$$

We have $\gamma_1(s)=\Gamma(s,0)$ and $\gamma_0(s)=\Gamma(s,1)$. Now, using as earlier the notation $S(s,t):=\partial_s\Gamma(s,t),\quad T(s,t):=\partial_t\Gamma(s,t)$, we have

$$\ell(t)=\int_0^1\left\langle S(s,t),S(s,t)\right\rangle^{\frac{1}{2}}ds \quad\text{and}\quad \frac{d\ell}{dt}(t)=\int_0^1\frac{\partial_t\left\langle S(s,t),S(s,t)\right\rangle}{2\left\|S(s,t)\right\|}ds.$$

Let $f(t)=\frac{1}{2}\partial_t\left\langle S(s,t),S(s,t)\right\rangle=\left\langle D_tS(s,t),S(s,t)\right\rangle$. We differentiate with respect to t and apply the Jacobi equation together with the definition of sectional curvature and obtain

$$\begin{aligned}\frac{df}{dt}(t)&=\left\langle D_t^2S(s,t),S(s,t)\right\rangle+\left\|D_tS(s,t)\right\|^2\\&=-\left\langle R\big(S(s,t),T(s,t)\big)T(s,t),S(s,t)\right\rangle+\left\|D_tS(s,t)\right\|^2\qquad(4.3.5)\\&=-K(s,t)\left(\|S\|^2\|T\|^2-\langle S,T\rangle^2\right)+\left\|D_tS(s,t)\right\|^2.\end{aligned}$$

Here R is the Riemannian curvature tensor and K is the sectional curvature. Since by assumption $K(s,t)\le 0$ it follows that $\frac{df}{dt}(t)\ge 0$ for $t\in[0,1]$. By the symmetry lemma [33, Lemma 6.2], we get

$$D_tS(s,t)\Big|_{t=0}=D_sT(s,0)=-hD_s\,X|_{\gamma_1(s)}.$$

Then

$$f(0)=-h\left\langle D_s\,X|_{\gamma_1(s)},S(s,0)\right\rangle\ge 0,$$

since X satisfies the monotonicity condition with $\nu=0$. So, we have

$$f(t)=f(0)+\int_0^t\frac{df}{d\tau}(\tau)\,d\tau\ge 0,$$

which allows us to conclude that $\frac{d\ell}{dt}(t) \geq 0$. Thus

$$\text{length}\left(\gamma_1(s)\right) = \ell(0) \leq \ell(1) = \text{length}\left(\gamma_0(s)\right). \tag{4.3.6}$$

For any given $\varepsilon > 0$, we have $\ell(1) < d\left(\gamma_0(0), \gamma_0(1)\right) + \varepsilon$. By (4.3.6) and the definition of distance we obtain

$$d\left(\gamma_1(0), \gamma_1(1)\right) \leq \ell(0) \leq \ell(1) \leq d\left(\gamma_0(0), \gamma_0(1)\right) + \varepsilon.$$

Since ε is arbitrary, the condition for B-stability is satisfied. □

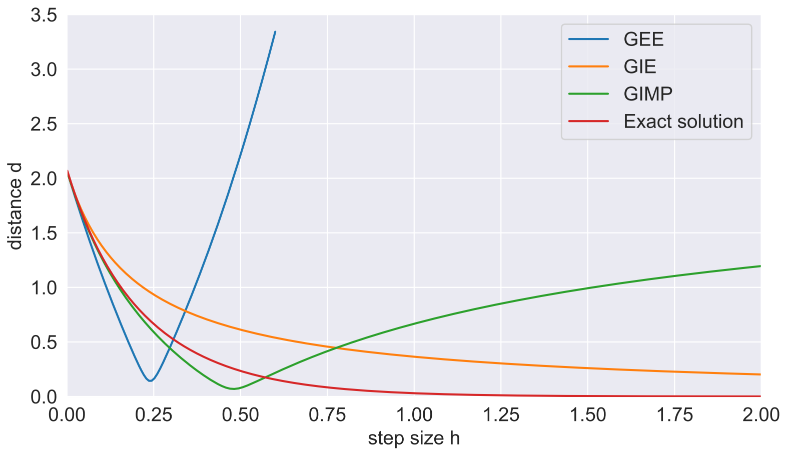

Figure 4.2: Riemannian distance of two solutions after one step plotted for increasing values of the step size h with the same initial values.

Example 11. *[$\mathbb{S}^n_{++}$] The space $\mathbb{S}^n_{++}$ of symmetric positive definite matrices is a well-known example of a manifold with negative sectional curvature. Its tangent space at a point A, denoted by $T_A\mathbb{S}^n_{++}$, can be identified as the set of $n \times n$ symmetric matrices. $\mathbb{S}^n_{++}$ is equipped pointwise with the metric*

$$g_A(U, V) = trace\left(A^{-1}UA^{-1}V\right), \tag{4.3.7}$$

where $A \in \mathbb{S}^n_{++}$ and $U, V \in T_A\mathbb{S}^n_{++}$, [43, 45].

The manifold $\mathbb{S}^n_{++}$ can be used as a model space for simple beam models, such as the Elastica [51], or in diffusion tensor magnetic resonance imaging (DT-MRI) [8, 13, 22, 44], via 3D tensors, i.e., 3×3 SPD matrices. Another interesting application is the segmentation and recognition of images and videos represented by SPD matrices, [1, 29, 49]. Such applications usually involve averaging SPD matrices, for example, to collect noisy measurements of the object under consideration. In $\mathbb{S}^n_{++}$, a suitable mean was proposed *by Karcher [31]. Given k matrices $Y_1,\dots,Y_k \in \mathbb{S}^n_{++}$, we search for a matrix $X^* \in \mathbb{S}^n_{++}$, the Karcher mean, such that*

$$X^* = \underset{X\in\mathbb{S}^n_{++}}{\arg\min} \frac{1}{2}\sum_{j=1}^{k} d^2\left(X, Y_j\right), \tag{4.3.8}$$

i.e., X^* is such that grad$\frac{1}{2}\sum_{j=1}^{k} d^2\left(X, Y_j\right) = 0$. Here, $d\left(X,Y\right)$ is the Riemannian distance between X and Y given as [23]

$$d\left(X,Y\right) = \sqrt{\sum_{i=1}^{n} \log^2\left(\lambda_i\left(X^{-\frac{1}{2}}YX^{-\frac{1}{2}}\right)\right)}, \tag{4.3.9}$$

with $\lambda_i\left(X^{-\frac{1}{2}}YX^{-\frac{1}{2}}\right)$ being the i^{th} eigenvalue of $X^{-\frac{1}{2}}YX^{-\frac{1}{2}}, i = 1,\dots,n$, and grad is the Riemannian gradient found e.g. in [23, Lemma 2]

$$\operatorname{grad}\frac{1}{2}d^2\left(X,Y\right)|_X = -X^{\frac{1}{2}}\log\left(X^{-\frac{1}{2}}YX^{-\frac{1}{2}}\right)X^{\frac{1}{2}}. \tag{4.3.10}$$

For $\mathbb{S}^{++}_n$, the exponential map is explicitly known in terms of the matrix exponential and matrix square roots as

$$\exp_A\left(tV\right) = A^{\frac{1}{2}}e^{tA^{-\frac{1}{2}}VA^{-\frac{1}{2}}}A^{\frac{1}{2}},$$

for $A \in \mathbb{S}^n_{++}$ and $V \in T_A\mathbb{S}^n_{++}$. The objective function $f\left(X\right) = \frac{1}{2}\sum_{j=1}^{k} d^2\left(X, Y_j\right)$ is defined as the geometric mean of symmetric positive definite matrices in [40] and [4], and is known to have a unique minimizer X^* as in (4.3.8)*, [31]. There is no known closed-form solution for* (4.3.8) *and usually, iterative methods are used to compute the Karcher mean.*
In Figure 4.2, the Riemannian distance of two solutions after one step is plotted for increasing values of the step size h with the same pair of initial values. One can observe the non-expansive behavior of the GIE and the GIMP method and the expansive behavior of the Geodesic Explicit Euler (GEE) method. The GEE solution is discontinued at h = 0.6 *for presentation purposes. The exact solution is calculated with strict tolerance by* `odeint` *of* `scipy.integrate` *in* `Python`*.*

4.3.2 The case with positive sectional curvature: The 2-sphere

In this section, we consider systems on the 2-sphere S^2 with the standard metric. We show through an example that the GIE and GIMP methods fail to be B-stable.

4.3.2.1 Killing vector fields

A *Killing vector field* is a vector field X such that the Lie derivative $\mathcal{L}_X g = 0$. This implies that

$$\begin{aligned} 0 &= (\mathcal{L}_X g)(Y,Z) = X\langle Y,Z\rangle - \langle \mathcal{L}_X Y, Z\rangle - \langle Y, \mathcal{L}_X Z\rangle \\ &= \langle \nabla_X Y, Z\rangle + \langle Y, \nabla_X Z\rangle - \langle \nabla_X Y - \nabla_Y X, Z\rangle - \langle Y, \nabla_X Z - \nabla_Z X\rangle \\ &= \langle \nabla_Y X, Z\rangle + \langle \nabla_Z X, Y\rangle, \end{aligned}$$

so that the monotonicity condition (4.2.1) holds with $\nu = 0$ for any such vector field. In this sense one could say that the Killing vector fields represent a borderline case for non-expansive systems.

4.3.2.2 A Killing vector field on S^2

Consider the vector field $X(y) = e_3 \times y$, which describes rotations on the 2-sphere around the z-axis. Using Cartesian coordinates, the GIE method (4.3.2) on the 2-sphere takes the form

$$y_0 = \exp_{y_1}\left(-h\, X|_{y_1}\right) = \cos\alpha \cdot y_1 - \frac{\sin\alpha}{\alpha}\cdot\left(h\, X|_{y_1}\right), \quad \alpha = \left\| -h\, X|_{y_1}\right\|. \tag{4.3.11}$$

We apply (4.3.11) to two initial points lying on the open northern hemisphere and measure the distance between the points for increasing values of the time step. The distance between two points $y_0, z_0 \in S^2$ is calculated as

$$d(y_0, z_0) = \arccos(y_0 \cdot z_0). \tag{4.3.12}$$

Figure 4.3 (left) shows one step performed with the GIE method starting from

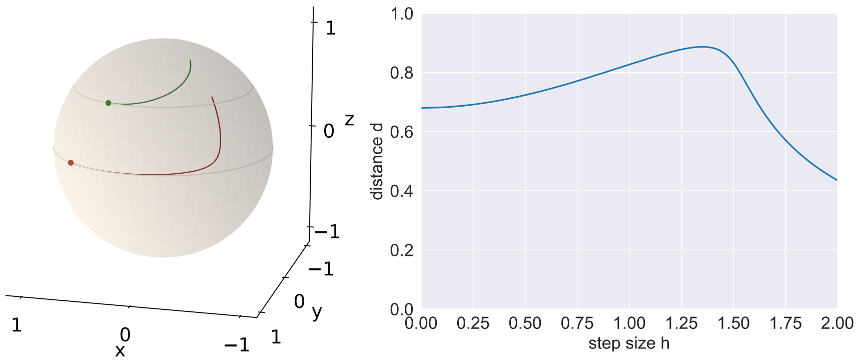

Figure 4.3: One step of GIE method for two initial points with increasing step size h (left) and their Riemannian distance (right).

two initial points with increasing step size h. In Figure 4.3 (right), the distance between the trajectories is shown as a function of h. As can be seen from the distance curve, the GIE method shows an expansive behavior, and it is in fact small values of the step size that cause problems. In Figure 4.4, the SPHMP and the GIMP methods are tested on the same vector field. Both methods are a reparametrization of the exact solution for this problem.

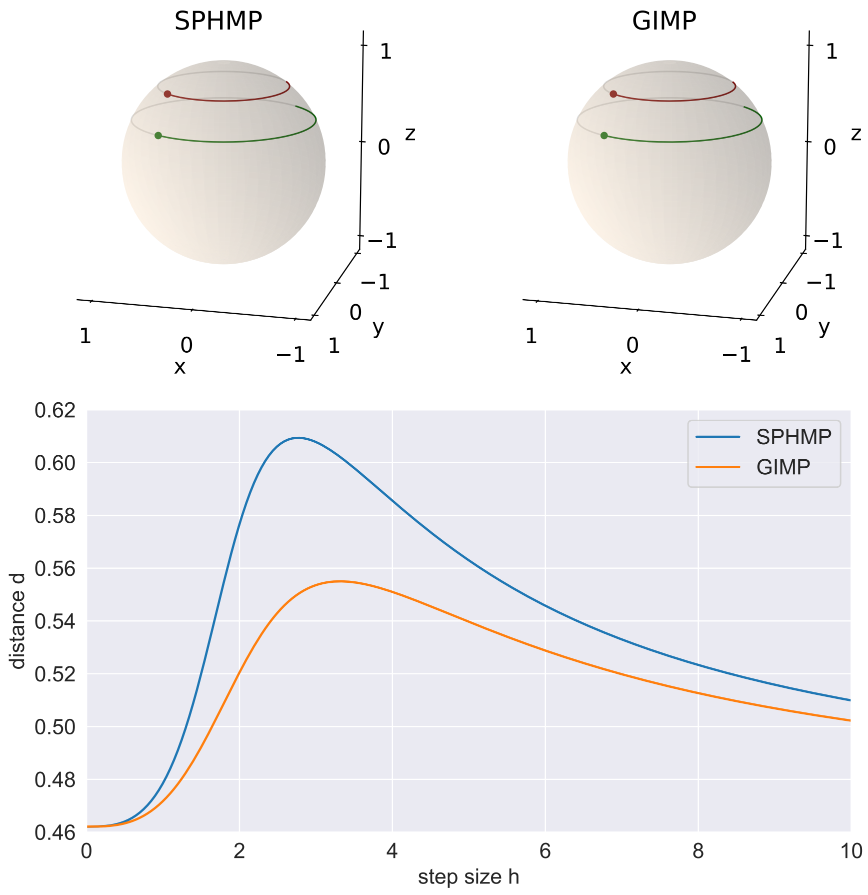

Figure 4.4: Top: One step of SPHMP (4.3.4) (left) and GIMP (4.3.3) (right) method for the same two initial points with increasing step size h. Bottom: Riemannian distance of two numerical solutions after one step plotted for increasing values of the step size h.

A non-uniqueness issue It is well-known from the theory of implicit Runge–Kutta methods that the conditions for the uniqueness of the solution to the implicit equations that must be solved in each time step involve the one-sided Lipschitz condition. In the monograph by Hairer and Wanner [24] a precise result is given, and we include it here for completeness.

Theorem 4.3 (Theorem 14.4 in [24])**.** *Consider a differential equation satisfying a one-sided Lipschitz condition with constant ν. If the Runge–Kutta matrix A is invertible and $h\nu < \alpha_0\left(A^{-1}\right)$, then the system of equations to be solved in each time step possesses at most one solution.*

We note that α_0 is a function that depends only on the Runge–Kutta coefficients, and it is known that $\alpha_0(A^{-1}) = 1$ for the implicit Euler method. Thus, for $\nu \le 0$ there is a unique solution for every $h > 0$. But the Killing vector field example on S^2 shows that this result is not generally true in Riemannian manifolds. In fact, for this example, we see from (4.3.11) that the last component is decoupled from the other two. Writing for simplicity $y_0^3 =: z_0$ and $y_1^3 =: z$ we need to solve the scalar equation

$$z_0 = \cos\left(h\sqrt{1-z^2}\right)z =: q\left(z,h\right) \qquad (4.3.13)$$

with respect to z. One has $q\left(0,h\right) = 0$ for all h, and $q\left(z_k,h\right) = 0$ for $z_k = \pm\sqrt{1-\left(\frac{\pi}{h}\right)^2\left(\frac{1}{2}+k\right)^2}$ for any $k \in \mathbb{N}$ such that $\left(\frac{\pi}{h}\right)^2\left(\frac{1}{2}+k\right)^2 \le 1$. In fact, for $h \in I_0 = \left(0, \frac{\pi}{2}\right]$, $q\left(z,h\right)$ has precisely one zero, and for $h \in I_m = \left((2m-1)\frac{\pi}{2}, (2m+1)\frac{\pi}{2}\right]$, $m \ge 1$, $q\left(z,h\right)$ has $2m+1$ zeros in $\left[-1,1\right]$. All the zeros are simple and therefore there is a sign change in $q(z,h)$ at each of them. It follows that $\exists\epsilon > 0$ such that if $h \in I_m$ and $\left|z_0\right| < \epsilon$, then (4.3.13) has at least $2m+1$ solutions. One easily verifies that for each of these values of the last component, there is a unique solution for the first two components. We illustrate the structure of the solution in Figure 4.5.

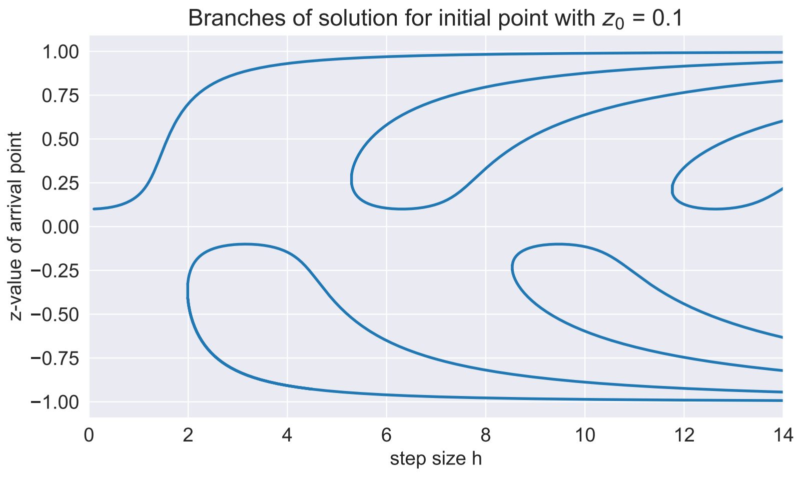

Figure 4.5: A bifurcation diagram for solutions to the equation (4.3.13).

4.3.2.3 Relation to other Lie group integrators

For some homogeneous manifolds $M = G/H$, with H a closed Lie subgroup of the Lie group G, the Geodesic Implicit Euler method (4.3.2) is equivalent to the implicit Lie-Euler method for a specific choice of isotropy, [11,41], i.e., of the map $a : G/H \to \mathfrak{g}$ which is used to define the Lie group method:

$$y_{n+1} = \exp\left(h a\left(y_{n+1}\right)\right) y_n, \tag{4.3.14}$$

with $\mathfrak{g}$ the Lie algebra of G and $\exp : \mathfrak{g} \to G$ the Lie group exponential. See [30] for an introduction to Lie group methods. The following example on S^2 illustrates the impact of the choice of isotropy on the approximation of the solution obtained via (4.3.14).

Example 12. *Consider a vector field on the 2-sphere $S^2 = SO(3)/SO(2)$. In Cartesian coordinates, embedding S^2 in $\mathbb{R}^3$, the ODE can be written as*

$$\dot{y} = a\left(y\right) \times y, \tag{4.3.15}$$

where $\times$ denotes the vector cross product. By the identification of $\left(\mathbb{R}^3, \times\right)$ with the Lie algebra $\mathfrak{so}(3)$, we have that $a : S^2 \to \mathbb{R}^3 \simeq \mathfrak{so}(3)$. The action of the Lie group exponential $\exp : \mathbb{R}^3 \simeq \mathfrak{so}(3) \to SO(3)$ *on a vector $p \in \mathbb{R}^3$ takes the simple form:*

$$\exp(a)\, p = p + \frac{\sin(\alpha)}{\alpha} a \times p - \frac{1-\cos(\alpha)}{\alpha^2} a \times \left(a \times p\right), \quad \alpha = \|a\|, \quad a \in \mathbb{R}^3 \simeq \mathfrak{so}(3).$$

We remark that for a given vector field $X\left(y\right) = a\left(y\right) \times y$, the choice of $a\left(y\right)$ is not unique. In fact, we can replace $a\left(y\right)$ with its projection orthogonal to y without changing $X\left(y\right)$, and similarly replacing $a\left(y\right)$ by $a\left(y\right) + c\left(y\right) y$, with $c : S^2 \to \mathbb{R}$, does not alter $X\left(y\right)$:

$$\dot{y} = a\left(y\right) \times y = \left(a\left(y\right) + c\left(y\right) y\right) \times y, \qquad y^\top a\left(y\right) = 0.$$

On the other hand, the numerical approximation obtained by the method (4.3.14),

$$y_{n+1} = \exp\left(h\left(a\left(y_{n+1}\right) + c\left(y_{n+1}\right) y_{n+1}\right)\right) y_n,$$

does depend on the choice of $c\left(y\right)$, see also Figure 4.6. Similarly, we cannot expect that in general different Lie group integrators have the same stability behavior when applied to the same vector field X.
In Figure 4.6 we illustrate the isotropy issue by applying the Implicit Lie–Euler method to the problem

$$\dot{y} = e_3 \times y = \left(e_3 + (c-1)\, y_3\, y\right) \times y \tag{4.3.16}$$

for $c \in [-2,2]$ and step size $h = 2$. This means that $c = 0$ corresponds to the GIE method, whereas for $c = 1$ the exact solution is reproduced. We observe that the difference in solutions may expand, contract or stay constant, depending on the choice of isotropy parameter c.

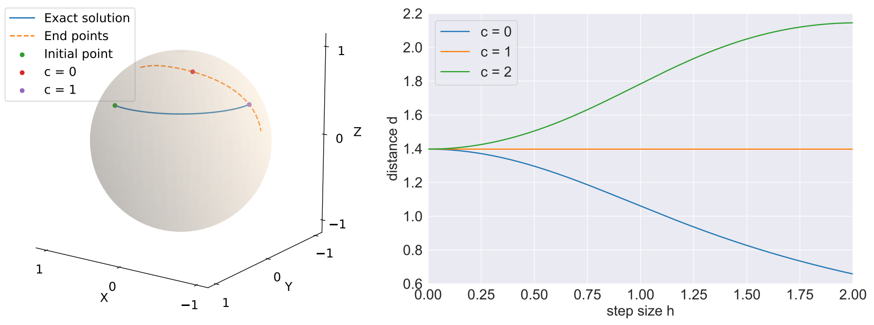

Figure 4.6: Left: The Implicit Lie-Euler method applied to (4.3.16) with stepsize $h = 2$ and $c \in [-2,2]$. The dashed curve shows the arrival point parametrized by c. The solid line depicts the exact solution. Right: the distance between two solutions for increasing stepsizes with three different choices of isotropy parameter $c \in \{0,1,2\}$.

4.4 A bound for the global error

For the next result, we first consider an initial value problem on the finite-dimensional Riemannian manifold (M,g),

$$\begin{cases} \dot{y} = X|_y \\ y(0) = y_0 \in M \end{cases}, \tag{4.4.1}$$

where X is a smooth vector field, $y_0 \in M$ is the initial value. The following theorem is a generalization of Theorem 2 from [9], where we use the constant ν from the monotonicity condition rather than the operator norm of ∇X.

Theorem 4.4. *Let (M,g) be a Riemannian manifold and fix $y_0 \in M$. Let $\mathcal{U}_{y_0} \subset M$ be a geodesically convex set and X a vector field on M satisfying the monotonicity condition* (4.2.1) *on $\mathcal{U}_{y_0}$ with constant $\nu \in \mathbb{R}$. Let $y(t) = \exp(tX)\, y_0$ be defined and contained in $\mathcal{U}_{y_0}$ for $t \in [0,t^*]$, $t^* > 0$. Let $\phi_{h,X}$ be a numerical method $y_{j+1} = \phi_{h,X}(y_j)$, $j = 0,\dots,k-1$, well defined and contained in $\mathcal{U}_{y_0}$ for any h such that $0 < h \le h^* \le t^*$, $t^* = hk$, whose local error can be bounded for some $p \in \mathbb{N}$ and $C \in \mathbb{R}$ as*

$$d\Big(\exp(hX)\, y, \phi_{h,X}(y)\Big) \le C h^{p+1} \quad \textit{for all } y \in \mathcal{U}_{y_0}, h \in (0,h^*]. \tag{4.4.2}$$

Then, for all $k > 0$, the global error is bounded as

$$d\left(y\left(t^*\right), y_k\right) \leq \begin{cases} \frac{C}{\nu}\left(\mathrm{e}^{t^*\nu} - 1\right) h^p & \textit{for } \nu > 0 \\ C t^* h^p & \textit{for } \nu = 0, \\ \frac{C\mathrm{e}^{-\nu h}}{\nu}\left(\mathrm{e}^{t^*\nu} - 1\right) h^p & \textit{for } \nu < 0 \end{cases} \quad h \in \left(0, h^*\right]. \tag{4.4.3}$$

Proof. Let us denote the global error as $E_k := d\left(y\left(t^*\right), y_k\right)$. For $j = 0, \ldots, k-1$,

$$E_{j+1} \leq d\left(\exp\left(hX\right) y\left(jh\right), \exp\left(hX\right) y_j\right) + d\left(\exp\left(hX\right) y_j, \phi_{h,X}\left(y_j\right)\right) \tag{4.4.4}$$

$$\leq \mathrm{e}^{h\nu} d\left(y\left(jh\right), y_j\right) + d\left(\exp\left(hX\right) y_j, \phi_{h,X}\left(y_j\right)\right) \tag{4.4.5}$$

$$= \mathrm{e}^{h\nu} E_j + d\left(\exp\left(hX\right) y_j, \phi_{h,X}\left(y_j\right)\right)$$

$$\leq \mathrm{e}^{h\nu} E_j + C h^{p+1}. \tag{4.4.6}$$

(4.4.4) is the triangle inequality, where the first term is the error at jh propagated over one step and the second term is the local error. (4.4.5) is obtained via a Grönwall-type inequality of [32] for the first term. Using the local error estimate (4.4.2) for the second term we obtain the recursion in (4.4.6). Considering $t^* = hk$, $\nu \neq 0$ and summing over $j = 1, \ldots, k-1$, we obtain

$$E_k \leq C \frac{\mathrm{e}^{t^*\nu} - 1}{\mathrm{e}^{h\nu} - 1} h^{p+1}. \tag{4.4.7}$$

For $\nu > 0$, $\mathrm{e}^{\nu h} - 1 > \nu h$ and (4.4.7) becomes equivalent to the first estimate in (4.4.3). For $\nu < 0$, $1 - \mathrm{e}^{-\nu h} < \nu h$ and (4.4.7) becomes equivalent to the third estimate in (4.4.3). □

Remark 6. *In cases where the monotonicity constant $\nu \ll 0$, in the sense that one can assume $\nu h \to -\infty$ as $h \to 0$, see e.g. [24, Ch IV.15], one gains an order of convergence such that the global error essentially equals the local error.*

4.5 Conclusions and further work

The notion of B-stability proposed in [6] for Euclidean spaces has been generalized to Riemannian manifolds. Building on the work by Simpson-Porco and Bullo [47] on contraction systems in Riemannian manifolds, we expressed the B-stability condition in terms of the Riemannian distance function. For this first study, only geodesic versions of the implicit Euler method and the implicit midpoint rule were considered. We proved that in the Riemannian setting, the geodesic implicit Euler method is B-stable for manifolds of non-positive sectional curvature, but not necessarily in positively curved spaces. Through numerical experiments on the 2-sphere, one finds strong evidence that the GIE method is indeed not B-stable in general. Another observation was that, contrary to what has been proved in Euclidean spaces, the nonlinear equations associated with the GIE method do not have a unique solution for non-expansive systems. Finally, we showed that the monotonicity constant can be used to obtain improved global error estimates compared to [9,16].

Many open questions remain for the B-stability properties of numerical methods applied to problems on Riemannian manifolds. There exist many classes of numerical integrators that could be analyzed in this setting. In mechanical engineering, most of the problems of interest are set in manifolds of positive sectional curvature, such as $SO(d), SE(d)$ $d = 2,3$, S^2, TS^2 and direct or semidirect products of these. It may also be of interest to consider explicit integrators, in which case B-stability must be replaced by some conditional form of stability, such as the circle contractivity proposed in [18].

Acknowledgements This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 860124. This publication reflects only the author's view and the Research Executive Agency is not responsible for any use that may be made of the information it contains.

Bibliography

[1] O. Arandjelovic, G. Shakhnarovich, J. Fisher, R. Cipolla, and T. Darrell, *Face recognition with image sets using manifold density divergence*, Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, IEEE, 2005, pp. 581–588.

[2] M. Arnold and O. Brüls, *Convergence of the generalized-α scheme for constrained mechanical systems*, Multibody System Dynamics **18** (2007), no. 2, 185–202.

[3] M. Arnold, O. Brüls, and A. Cardona, *Error analysis of generalized-α Lie group time integration methods for constrained mechanical systems*, Numerische Mathematik **129** (2015), no. 1, 149–179.

[4] R. Bhatia and J. Holbrook, *Riemannian geometry and matrix geometric means*, Linear algebra and its applications **413** (2006), no. 2-3, 594–618.

[5] G. Bogfjellmo and H. Marthinsen, *High-order symplectic partitioned Lie group methods*, Foundations of Computational Mathematics **16** (2016), no. 2, 493–530.

[6] J. C. Butcher, *A stability property of implicit Runge-Kutta methods*, BIT Numerical Mathematics **15** (1975), 358–361.

[7] E. Celledoni, E. Çokaj, A. Leone, D. Murari, and B. Owren, *Lie group integrators for mechanical systems*, International Journal of Computer Mathematics **99** (2022), no. 1, 58–88.

[8] E. Celledoni, S. Eidnes, B. Owren, and T. Ringholm, *Dissipative numerical schemes on Riemannian manifolds with applications to gradient flows*, SIAM Journal on Scientific Computing **40** (2018), no. 6, 3789–3806.

[9] ______, *Energy-preserving methods on Riemannian manifolds*, Mathematics of Computation **89** (2020), no. 322, 699–716.

[10] E. Celledoni, A. Marthinsen, and B. Owren, *Commutator-free Lie group methods*, Future Generation Computer Systems **19** (2003), 341–352.

[11] E. Celledoni, H. Marthinsen, and B. Owren, *An introduction to Lie group integrators - basics, new developments and applications*, Journal of Computational Physics **257** (2014), 1040–1061.

[12] E. Celledoni and B. Owren, *A class of intrinsic schemes for orthogonal integration*, SIAM Journal on Numerical Analysis **40** (2002), no. 6, 2069–2084.

[13] G. Cheng, H. Salehian, and B. C. Vemuri, *Efficient recursive algorithms for computing the mean diffusion tensor and applications to DTI segmentation*, Computer Vision-ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VII 12, Springer, 2012, pp. 390–401.

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

[15] P. E. Crouch and R. Grossman, *Numerical integration of ordinary differential equations on manifolds*, Journal of Nonlinear Science **3** (1993), 1–33.

[16] C. Curry and A. Schmeding, *Convergence of Lie group integrators*, Numerische Mathematik **144** (2020), no. 2, 357–373.

[17] G. Dahlquist, *Error analysis for a class of methods for stiff nonlinear initial value problems, in Lecture Notes in Math. 506, G.A. Watson (ed.)*, Springer-Verlag, Berlin (1976).

[18] G. Dahlquist and R. Jeltsch, *Generalized disks of contractivity for explicit and implicit Runge-Kutta methods*, Dept. of Numerical Analysis and Computer Science, The Royal Institute of Technology, Stockholm, Report TRITA-NA-7906 (1979).

[19] A. Davydov, S. Jafarpour, and F. Bullo, *Non-Euclidean contraction theory for robust nonlinear stability*, Institute of Electrical and Electronics Engineers. Transactions on Automatic Control **67** (2022), no. 12, 6667–6681.

[20] K. Dekker and J. G. Verwer, *Stability of Runge-Kutta methods for stiff nonlinear differential equations*, North-Holland, Amsterdam-New-York-Oxford, 1984.

[21] F. Demoures, F. Gay-Balmaz, S. Leyendecker, S. Ober-Blöbaum, T. S. Ratiu, and Y. Weinand, *Discrete variational Lie group formulation of geometrically exact beam dynamics*, Numerische Mathematik **130** (2015), no. 1, 73–123.

[22] P. T. Fletcher and S. Joshi, *Riemannian geometry for the statistical analysis of diffusion tensor data*, Signal Processing **87** (2007), no. 2, 250–262.

[23] R. M. Gregório and P. R. Oliveira, *A proximal technique for computing the Karcher mean of symmetric positive definite matrices*, Optimization Online (2013).

[24] E. Hairer and G. Wanner, *Solving ordinary differential equations II, Stiff and differential-algebraic problems*, Second revised edition ed., Springer-Verlag, 1996.

[25] J. Hall and M. Leok, *Lie group spectral variational integrators*, Foundations of Computational Mathematics **17** (2017), 199–257.

[26] S. Hante and M. Arnold, *RATTLie: A variational Lie group integration scheme for constrained mechanical systems*, Journal of Computational and Applied Mathematics **387** (2021), 112492.

[27] 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** (1977), no. 3, 283–292.

[28] S. Holzinger and J. Gerstmayr, *Time integration of rigid bodies modelled with three rotation parameters*, Multibody System Dynamics (2021), 1–34.

[29] Z. Huang, R. Wang, S. Shan, and X. Chen, *Face recognition on large-scale video in the wild with hybrid Euclidean-and-Riemannian metric learning*, Pattern Recognition **48** (2015), no. 10, 3113–3124.

[30] A. Iserles, H. Munthe-Kaas, S. P. Nørsett, and A. Zanna, *Lie Group Methods*, Acta Numerica **9** (2000), 215–365.

[31] H. Karcher, *Riemannian center of mass and mollifier smoothing*, Communications on pure and applied mathematics **30** (1977), no. 5, 509–541.

[32] M. Kunzinger, H. Schichl, R. Steinbauer, and J. A. Vickers, *Global Gronwall estimates for integral curves on Riemannian manifolds*, Revista Matemática Complutense **19** (2006), no. 1, 133–137.

[33] J. M. Lee, *Introduction to Riemannian manifolds*, Graduate Texts in Mathematics, vol. 176, Springer, Cham, 2018.

[34] T. Lee, M. Leok, and N. H. McClamroch, *Lie group variational integrators for the full body problem*, Computer Methods in Applied Mechanics and Engineering **196** (2007), no. 29-30, 2907–2924.

[35] B. Leimkuhler and G. W. Patrick, *A symplectic integrator for Riemannian manifolds*, Journal of Nonlinear Science **6** (1996), no. 4, 367–384.

[36] T. Leitz and S. Leyendecker, *Galerkin Lie-group variational integrators based on unit quaternion interpolation*, Computer Methods in Applied Mechanics and Engineering **338** (2018), 333–361.

[37] D. Lewis and J. C. Simo, *Conserving algorithms for the dynamics of Hamiltonian systems on Lie groups*, Journal of Nonlinear Science **4** (1994), 253–299.

[38] J. E. Marsden and M. West, *Discrete mechanics and variational integrators*, Acta Numerica **10** (2001), 357–514.

[39] R. McLachlan, K. Modin, and O. Verdier, *A minimal-variable symplectic integrator on spheres*, Mathematics of Computation **86** (2017), no. 307, 2325–2344.

[40] M. Moakher, *A differential geometric approach to the geometric mean of symmetric positive-definite matrices*, SIAM journal on matrix analysis and applications **26** (2005), no. 3, 735–747.

[41] H. Munthe-Kaas, *High order Runge-Kutta methods on manifolds*, Applied Numerical Mathematics **29** (1999), 115–127.

[42] B. Owren, *Lie group integrators*, In: K. Ebrahimi-Fard and M. Barbero Liñán (eds.): Discrete mechanics, geometric integration and Lie-Butcher series, Springer Proc. Math. Stat., vol. 267, Springer, Cham, 2018, pp. 29–69. MR 3883642

[43] X. Pennec, P. Fillard, and N. Ayache, *A Riemannian framework for tensor computing*, International Journal of computer vision **66** (2006), 41–66.

[44] Y. Rathi, A. Tannenbaum, and O. Michailovich, *Segmenting images on the tensor manifold*, 2007 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2007, pp. 1–8.

[45] C. L. Siegel, *Symplectic geometry*, 1964.

[46] J. C. Simo and L. Vu-Quoc, *On the dynamics of finite-strain rods undergoing large motions – A geometrically exact approach*, Computer Methods in Applied Mechanics and Engineering **66** (1988), 125–161.

[47] J. W. Simpson-Porco and F. Bullo, *Contraction theory on Riemannian manifolds*, Systems & Control Letters **65** (2014), 74–80.

[48] Z. Terze, A. Müller, and D. Zlatar, *Singularity-free time integration of rotational quaternions using non-redundant ordinary differential equations*, Multibody System Dynamics **38** (2016), no. 3, 201–225.

[49] O. Tuzel, F. Porikli, and P. Meer, *Region covariance: A fast descriptor for detection and classification*, Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part II 9, Springer, 2006, pp. 589–600.

[50] A. Zanna, K. Engø, and H. Munthe-Kaas, *Adjoint and selfadjoint Lie-group methods*, BIT Numerical Mathematics **41** (2001), no. 2, 395–421.

[51] E. Zhang and L. Noakes, *Riemannian cubics and elastica in the manifold SPD(n) of all $n \times n$ symmetric positive-definite matrices*, Journal of Geometric Mechanics **11** (2019), no. 2, 277–299.

Part II

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 to Multibody System Dynamics.

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

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)
$q_{\boldsymbol{\rho}}^{\mathrm{d}}$	discrete neural network
$q_{\boldsymbol{\rho}}^{\mathrm{c}}$	continuous neural network approximating the solution curve $\boldsymbol{q}(s)$
$\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$	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.

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 $\mathbf{q} \in C^2\left(\left[0, L\right], \mathbb{R}^2\right)$, i.e., the curve is planar and twice continuously differentiable with length L. If s denotes the arc length parameter, then $\|\mathbf{q}'(s)\| = 1$, where $' = \frac{d}{ds}$, for all $s \in \left[0, L\right]$. 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 $\kappa(s) = \|\mathbf{q}''(s)\|$.

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 $\mathbb{R}^2$:

$$\mathcal{L}\left(\mathbf{q}, \mathbf{q}', \mathbf{q}''\right) = \frac{1}{2} EI \|\mathbf{q}''\|^2 . \tag{5.2.1}$$

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 $\widetilde{\mathcal{L}} : T^{(2)}Q \times \mathbb{R} \to \mathbb{R}$

$$\widetilde{\mathcal{L}}\left(\mathbf{q},\mathbf{q}',\mathbf{q}'',\Lambda\right) = \mathcal{L}\left(\mathbf{q},\mathbf{q}',\mathbf{q}''\right) + \Lambda\Phi\left(\mathbf{q},\mathbf{q}'\right), \tag{5.2.3}$$

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:

$$\mathcal{S}[\mathbf{q}] = \int_0^L \widetilde{\mathcal{L}}\left(\mathbf{q},\mathbf{q}',\mathbf{q}'',\Lambda\right) ds. \tag{5.2.4}$$

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 $\mathbf{q}(s_k)$, $\mathbf{q}'(s_k)$, and $\Lambda(s_k)$, and the curvature on the interval $[s_k, s_{k+1}]$ is approximated in terms of lower order derivatives as follows

$$\begin{aligned}
\mathbf{q}''(s_k) \approx \left(\mathbf{q}''_k\right)^- &= \frac{\left(-2\mathbf{q}'_{k+1} - 4\mathbf{q}'_k\right)h + 6\left(\mathbf{q}_{k+1} - \mathbf{q}_k\right)}{h^2}, \\
\mathbf{q}''(s_{k+1}) \approx \left(\mathbf{q}''_{k+1}\right)^+ &= \frac{\left(4\mathbf{q}'_{k+1} + 2\mathbf{q}'_k\right)h - 6\left(\mathbf{q}_{k+1} - \mathbf{q}_k\right)}{h^2}.
\end{aligned}$$

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 $j = 1,\ldots,\ell$, 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 $\mathrm{Loss}\left(\boldsymbol{\rho}\right)$.

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 $\|\cdot\|$

$$\mathrm{Loss}(\boldsymbol{\rho}) = \frac{1}{M}\sum_{i=1}^{M}\left\|f_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right) - \mathbf{y}^i\right\|^2.$$

The training of the network is the process of minimising $\mathrm{Loss}(\boldsymbol{\rho})$ with respect to $\boldsymbol{\rho}$ and it is usually done with gradient descent (GD):

$$\boldsymbol{\rho}^{(k)} \mapsto \boldsymbol{\rho}^{(k)} - \eta\nabla\mathrm{Loss}\left(\boldsymbol{\rho}^{(k)}\right) =: \boldsymbol{\rho}^{(k+1)}.$$

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

$$\mathbf{y}^i = \left(\hat{\mathbf{q}}_1^i, \dots, \hat{\mathbf{q}}_{N-1}^i, \left(\hat{\mathbf{q}}_1^i\right)', \dots, \left(\hat{\mathbf{q}}_{N-1}^i\right)'\right) \in \mathbb{R}^{4(N-1)}$$

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

$$\mathrm{Loss}(\boldsymbol{\rho}) = \frac{1}{4M(N-1)} \sum_{i=1}^{M} \left\| q_{\boldsymbol{\rho}}^{\mathrm{d}}\left(\mathbf{x}^i\right) - \mathbf{y}^i \right\|_W^2, \tag{5.4.1}$$

will be used to learn the input-to output map $q_{\boldsymbol{\rho}}^{\mathrm{d}} : \mathbb{R}^8 \to \mathbb{R}^{4(N-1)}$, 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.

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	$[1\cdot 10^{-4}, 1\cdot 10^{-1}]$	log uniform
weight decay	$[1\cdot 10^{-7}, 5\cdot 10^{-4}]$	log uniform
γ	$[0, 1\cdot 10^{-2}]$	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.

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 reported in Table 5.3. The resulting training error on the *both-end* data set is $2.791 \cdot 10^{-7}$, and the test error on a set of trajectories belonging to the same data set is $3.028 \cdot 10^{-7}$. 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.

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^{d}_{ρ} tested on the *both-ends* data set with 90% – 10% splitting into training and test set. The results are shown in Figure 5.1.

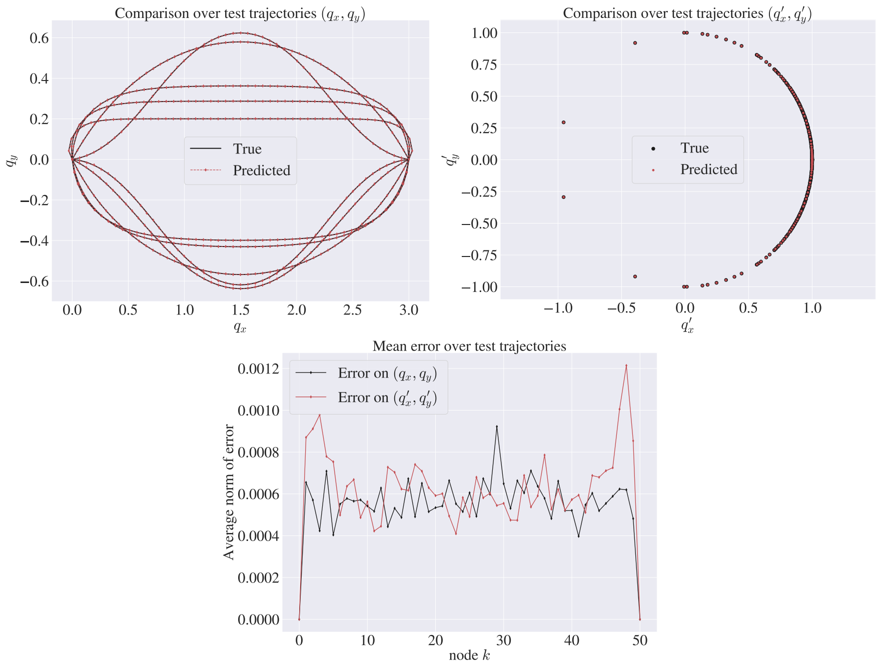

Figure 5.1: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network q^{d}_{ρ} tested on the *both-ends* data set with 90% – 10% splitting into training and test set. These results are obtained with the hyperparameters from Table 5.3, that yield a training error equal 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.

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.

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 $q_{\boldsymbol{\rho}}^{\mathrm{d}}$ 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.

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, $3.047 \cdot 10^{-7}$ and $3.141 \cdot 10^{-7}$. Finally, we remark that the test accuracy is a good measure of the generalisation error of neural network under the hypothesis that 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

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 $q_{\boldsymbol{\rho}}^{\mathrm{d}}$ 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.

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.

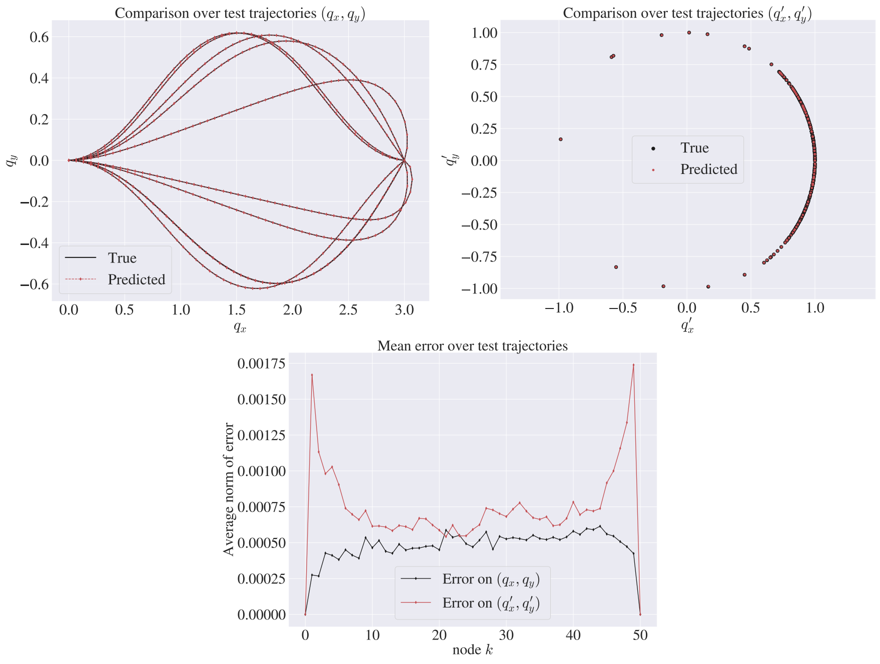

Figure 5.2: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the discrete network $q^{\mathrm{d}}_{\boldsymbol{\rho}}$ tested on the *both-ends* + *right-end* data set with $90\% - 10\%$ splitting into training and test set. These results are obtained with the hyperparameters 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.

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 $N+1$, 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 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

$$\Omega = \left\{ \left(s_k, \mathbf{x}^i \right), \mathbf{y}_k^i \right\}_{k=0,\dots,N}^{i=1,\dots,M},$$

where, as in the previous section,

$$\mathbf{x}^i = \left(\mathbf{q}_0^i, \mathbf{q}_N^i, \left(\mathbf{q}_0^i\right)', \left(\mathbf{q}_N^i\right)' \right) \in \mathbb{R}^8,$$

and

$$\mathbf{y}_k^i = \left(\hat{\mathbf{q}}_k^i, \left(\hat{\mathbf{q}}_k^i\right)' \right).$$

Here $\hat{\mathbf{q}}_k^i$ 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

$$q_{\boldsymbol{\rho}}^{\mathrm{c}} : \mathbb{R}^8 \to \mathcal{C}^\infty\left([0, L], \mathbb{R}^2 \right),$$

and the differential operator

$$\mathcal{D} : \mathcal{C}^\infty\left([0, L], \mathbb{R}^2 \right) \to \mathcal{C}^\infty\left([0, L], \mathbb{R}^2 \right), \; \mathcal{D}\left(q_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\mathbf{x}^i\right) \right)(s_k) = \frac{d}{ds}\left(q_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\boldsymbol{x}^i\right) \right)(s) \Big|_{s=s_k},$$

so that we can define

$$y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) := \left(q_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\mathbf{x}^i\right)(s_k), \mathcal{D}\left(q_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\mathbf{x}^i\right) \right)(s_k) \right).$$

To train the network $q_{\boldsymbol{\rho}}^c$, we define the loss function

$$\begin{aligned}\mathrm{Loss}(\boldsymbol{\rho}) = \frac{1}{4M(N+1)} \sum_{i=1}^{M} \sum_{k=0}^{N} \Bigg(& \left\| y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) - \boldsymbol{y}_k^i \right\|_2^2 \\ & + \gamma \left(\left\| \pi_{\mathcal{D}}\left(y_{\boldsymbol{\rho}}\left(\mathbf{x}^i\right)(s_k) \right) \right\|_2^2 - 1 \right)^2 \Bigg), \end{aligned} \qquad (5.5.1)$$

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.

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.

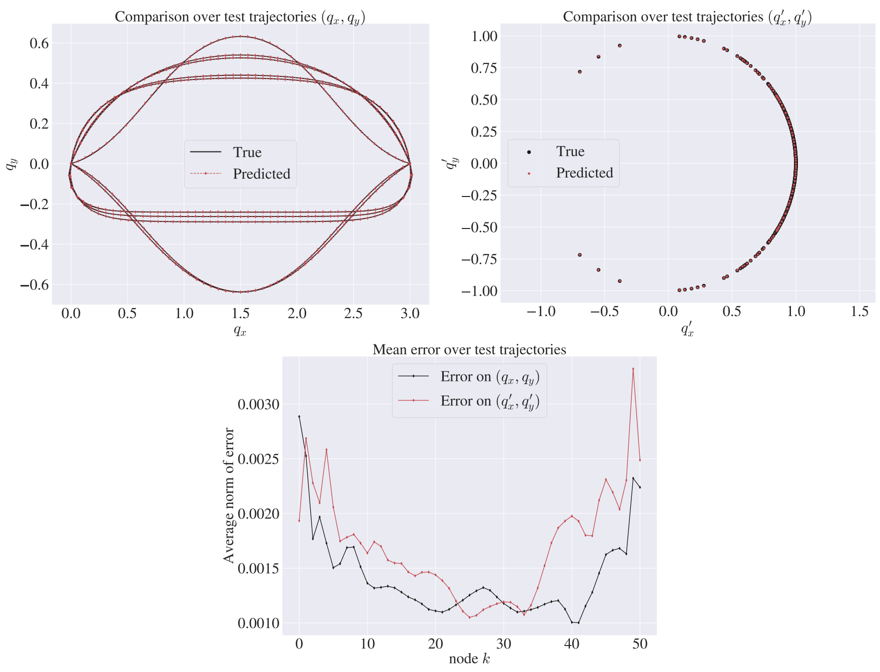

Figure 5.3: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$ for the continuous network $q^{\mathrm{c}}_{\boldsymbol{\rho}}$ tested on the *both-ends* data set with 90% – 10% splitting into training and test set. These results are obtained with the hyperparameters from Table 5.7, 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.

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.

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.

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.

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 $\mathbf{q}'(s) = \left(\cos\left(\theta(s)\right), \sin\left(\theta(s)\right)\right)$. 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:

$$\hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) = f_{\boldsymbol{\rho}}(s, \theta^i_0, \theta^i_N), \tag{5.5.6}$$

$$\begin{aligned} \hat{\theta}^{\mathrm{c}}_{\boldsymbol{\rho}}\left(\boldsymbol{x}^i\right)(s) = f_{\boldsymbol{\rho}}(s, \theta^i_0, \theta^i_N) + \left(\theta^i_0 - f_{\boldsymbol{\rho}}\left(0, \theta^i_0, \theta^i_N\right)\right) e^{-100 s^2} \\ + \left(\theta^i_N - f_{\boldsymbol{\rho}}\left(L, \theta^i_0, \theta^i_N\right)\right) e^{-100 (s-L)^2}, \end{aligned} \tag{5.5.7}$$

where $f_{\boldsymbol{\rho}} : \mathbb{R}^3 \to \mathbb{R}$ is any neural network, and we recall that $\pi\left(\boldsymbol{x}^i\right) = \left(\theta_0^i, \theta_N^i\right)$. We remark that, in the case of the parameterisation in Equation (5.5.7), one gets $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\boldsymbol{x}^i\right)(0) = \theta_0^i$ and $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}\left(\boldsymbol{x}^i\right)(L) = \theta_N^i$ 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	$\{3, \ldots, 8\}$	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 $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$ 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 $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$ 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,

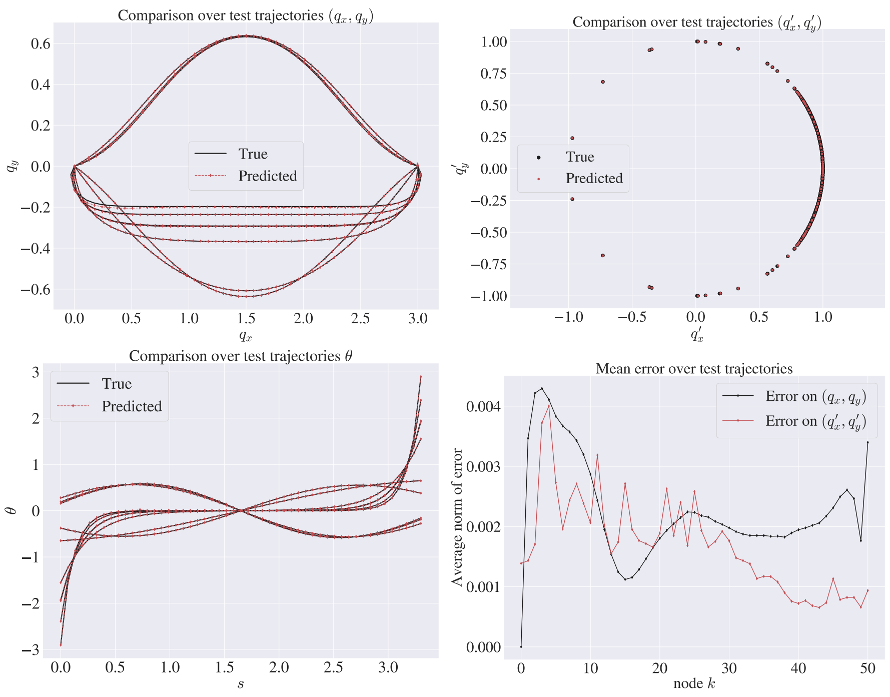

Figure 5.4: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta^{\mathrm{c}}_{\boldsymbol{\rho}}$ is modelled as in Equation (5.5.6), with 90% – 10% splitting of the *both-ends* data set into training and test set. These results are obtained with the hyperparameters 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.

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.

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 $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$. 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.

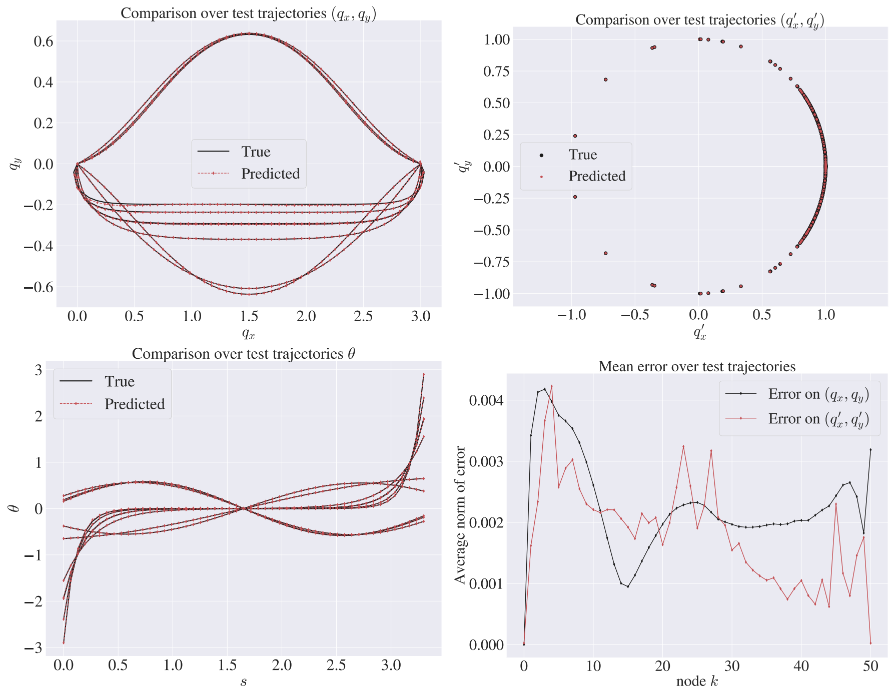

Figure 5.5: Comparison over test trajectories for $\mathbf{q}$ and $\mathbf{q}'$, for the case $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$ is modelled as in Equation (5.5.7), with $90\% - 10\%$ splitting of the *both-ends* data set into training and test set. These results are obtained with the hyperparameters 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.

Comparing the results related to $q_{\boldsymbol{\rho}}^{\mathrm{c}}$ with those of $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$, 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 $q_{\boldsymbol{\rho}}^{\mathrm{d}}$. 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 $q_{\boldsymbol{\rho}}^{\mathrm{c}}$ is more suitable for an easy parametrisation of both $\mathbf{q}$ and $\mathbf{q}'$, while $\theta_{\boldsymbol{\rho}}^{\mathrm{c}}$ is more suitable to impose geometrical structure and constraints.

The total accuracy error of a neural network model can be defined by splitting 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 equations (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.

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:

$$\mathbf{U} = \sigma\left(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1\right), \quad \mathbf{V} = \phi\left(\mathbf{W}_2\mathbf{x}+\mathbf{b}_2\right) \tag{5.A.2}$$

$$\mathbf{H}_1 = \sigma\left(\mathbf{W}_3\mathbf{x}+\mathbf{b}_3\right) \tag{5.A.3}$$

$$\mathbf{Z}_j = \sigma\left(\mathbf{W}_j^z\mathbf{H}_j+\mathbf{b}_j^z\right), \; j = 1,\ldots,\ell \tag{5.A.4}$$

$$\mathbf{H}_{j+1} = \left(1-\mathbf{Z}_j\right)\odot\mathbf{U}+\mathbf{Z}_j\odot\mathbf{V}, \; j = 1,\ldots,\ell \tag{5.A.5}$$

$$f_{\boldsymbol{\rho}}^{MULT}(\mathbf{x}) = \mathbf{W}\mathbf{H}_{\ell+1}+\mathbf{b}, \tag{5.A.6}$$

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.

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	$1.019\cdot10^{-3}$	$5.455\cdot10^{-3}$	$1.429\cdot10^{-3}$	$1.378\cdot10^{-3}$
weight decay	$1.79\cdot10^{-6}$	$2.142\cdot10^{-7}$	$1.317\cdot10^{-7}$	$1.535\cdot10^{-7}$
γ	$8.595\cdot10^{-3}$	$2.973\cdot10^{-3}$	$9.338\cdot10^{-3}$	$4.242\cdot10^{-3}$
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_{ρ}^{c} tested on the *both-ends* data set with different numbers of training data points. The weight decay is systematically set to 0.

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.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019, pp. 2623–2631.

[2] D. Bigoni, *Nonlinear solid mechanics: bifurcation theory and material instability*, Cambridge University Press, 2012.

[3] S. C. Brenner, *The mathematical theory of finite element methods*, Springer, 2008.

[4] S. L. Brunton and J. N. Kutz, *Machine learning for partial differential equations*, 2023, arXiv:2303.17078.

[5] S. Chevalier, J. Stiasny, and S. Chatzivasileiadis, *Accelerating Dynamical System Simulations with Contracting and Physics-Projected Neural-Newton Solvers*, Learning for Dynamics and Control Conference (PMLR), 2022, pp. 803–816.

[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** (2016), 1615–1650.

[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** (2022), no. 3, 88.

[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** (2022), no. 6, 063107.

[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.

[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** (2023), no. 4, 043128.

[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** (2021), no. 19, 321–326.

[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** (2023), no. 5, A2356–A2381.

[13] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations I, Nonstiff problems*, Second revised edition ed., Springer-Verlag, 1993.

[14] E. Hairer and G. Wanner, *Solving ordinary differential equations II, Stiff and differential-algebraic problems*, Second revised edition ed., Springer-Verlag, 1996.

[15] C. F. Higham and D. J. Higham, *Deep learning: An introduction for applied mathematicians*, Siam review **61** (2019), no. 4, 860–891.

[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** (2022), no. 2, 025002.

[17] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, International Conference on Learning Representations (ICLR) (San Diega, CA, USA), 2015.

[18] S. Kollmannsberger, D. D'Angella, M. Jokeit, and L. Herrmann, *Deep learning in computational mechanics*, Springer, 2021.

[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** (1998), no. 5, 987–1000.

[20] Y. Li, Z. Zhou, and S. Ying, *Delisa: Deep learning based iteration scheme approximation for solving PDEs*, Journal of Computational Physics **451** (2022), 110884.

[21] Y. Liu, J. N. Kutz, and S. L. Brunton, *Hierarchical deep learning of multiscale differential equation time-steppers*, 2020, arXiv:2008.09768.

[22] A. E. H. Love, *A treatise on the mathematical theory of elasticity*, Cambridge University Press, Cambridge, 1863 - 1940.

[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** (2021), no. 3, 218–229.

[24] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *Deepxde: A deep learning library for solving differential equations*, SIAM review **63** (2021), no. 1, 208–228.

[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 (2023), 1–16.

[26] J. E. Marsden and M. West, *Discrete mechanics and variational integrators*, Acta numerica **10** (2001), 357–514.

[27] S. Matsutani, *Euler's elastica and beyond*, Journal of Geometry and Symmetry in Physics **17** (2010), 45–86.

[28] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas, *Hamiltonian neural networks for solving equations of motion*, Physical Review E **105** (2022), no. 6, 065305.

[29] D. Mortari, H. Johnston, and L. Smith, *High accuracy least-squares solutions of nonlinear differential equations*, Journal of computational and applied mathematics **352** (2019), 293–307.

[30] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer, 1999.

[31] K. Ntarladima, M. Pieber, and J. Gerstmayr, *A model for contact and friction between beams under large deformation and sheaves*, Nonlinear Dynamics **111** (2023), no. 22, 20643–20660.

[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, vol. 32, Curran Associates, Inc., 2019.

[33] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*, vol. 37, Springer Science & Business Media, 2006.

[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** (2019), 686–707.

[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.

[36] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.

[37] M. A. Saadat and D. Durville, *A mixed stress-strain driven computational homogenization of spiral strands*, Computers & Structures **279** (2023), 106981.

[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** (2020), 112790.

[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** (2021), 334–356.

[40] D. A. Singer, *Lectures on elastic curves and rods*, AIP Conference Proceedings (American Institute of Physics), vol. 1002, 2008, pp. 3–32.

[41] M. Stavole, R. T. S. M. de Almagro, M. Lohk, and S. Leyendecker, *Homogenization of the constitutive properties of composite beam cross-sections*, ECCOMAS Congress 2022-8th European Congress on Computational Methods in Applied Sciences and Engineering, 2022.

[42] S. P. Timoshenko and J. M. Gere, *Theory of elastic stability*, McGraw-Hill Book Company, 1961.

[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** (2020), 261–272.

[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** (2023), no. 2, 1069–1343.

[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** (2021), no. 5, A3055–A3081.

[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** (2018), no. 1, 1–12.

[47] G. Yagawa and A. Oishi, *Computational mechanics with deep learning: An introduction*, Springer Nature, 2022.

Supervised Time Series Classification for Anomaly Detection in Subsea Engineering

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

Accepted for publication by 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 6.1 in Section 6.2.

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 [26] and Hotelling [12]. We use [13] 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 [4] 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. [10, 14, 21].

Section 6.5 covers Decision Trees (DTs), a popular supervised classification and regression technique introduced in the 1960s by Morgan and Sonquist in [23]. New concepts, reviews of decision trees and their applications in different fields such as medicine, finance, environmental sciences, are presented in [27, 34, 36].

Section 6.6 illustrates how to use a Support Vector Machine (SVM) [5], 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 [7].

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 [17], 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 [3, 9, 18].

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 [24]. 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 [37] 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 [37].

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 [32] 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 $X_{\text{uts}} = [x_1, x_2, \dots, x_n]$, which is an ordered set of real values x_t indexed by integers $t = 1, 2, \dots, n$, 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

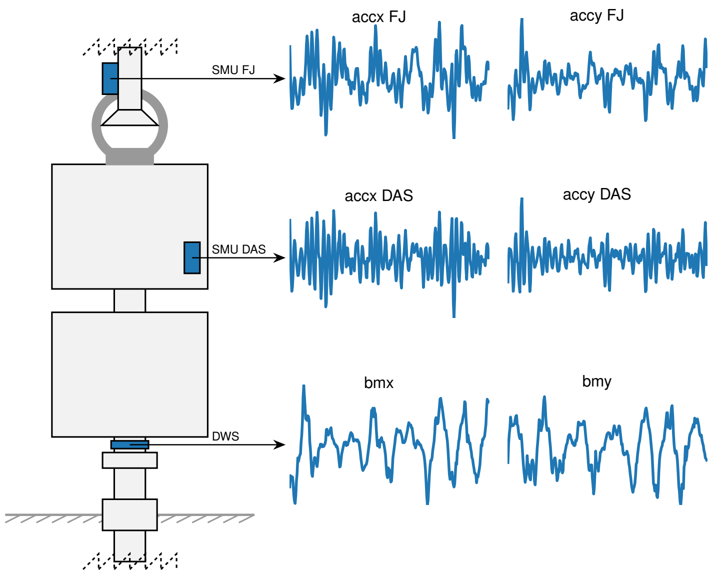

Figure 6.1: Stack with sensors and corresponding data

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

$$X_{\text{mts}} = \left[X^1_{\text{uts}}, X^2_{\text{uts}}, \dots, X^6_{\text{uts}}\right] \in \mathbb{R}^{n\times m}. \tag{6.2.1}$$

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.

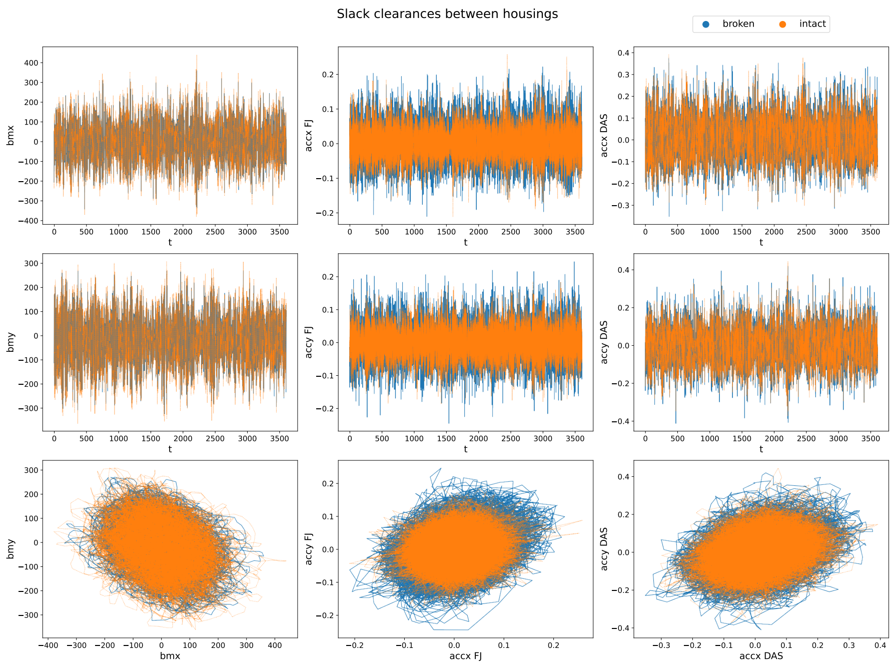

Figure 6.2: Two 1-hour simulations from the dataset comparing a broken and intact well under similar conditions. Plots are given for the x and y component of the different physical measurements. The two top rows give the time series while the bottom row shows phase plots.

6.2.1.1 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.

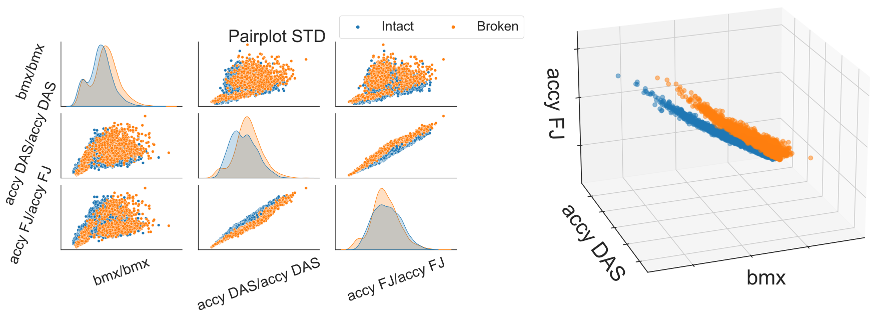

Figure 6.3: Pair plot showing of the scatter and distribution of data after a standard deviation transform (left). Plot visualizing the transformed data in 3 dimensions (right).

6.2.1.2 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 [33], therefore we introduce the covariance matrix

$$\Sigma = \begin{bmatrix} \operatorname{Var}(X_1) & \operatorname{Cov}(X_1, X_2) & \cdots & \operatorname{Cov}(X_1, X_n) \\ \operatorname{Cov}(X_2, X_1) & \operatorname{Var}(X_2) & \cdots & \operatorname{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{Cov}(X_n, X_1) & \operatorname{Cov}(X_n, X_2) & \cdots & \operatorname{Var}(X_n) \end{bmatrix}. \tag{6.2.3}$$

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

$$\operatorname{Cor}(X) = \operatorname{diag}(\Sigma)^{-\frac{1}{2}}\ \Sigma\ \operatorname{diag}(\Sigma)^{-\frac{1}{2}}. \tag{6.2.4}$$

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

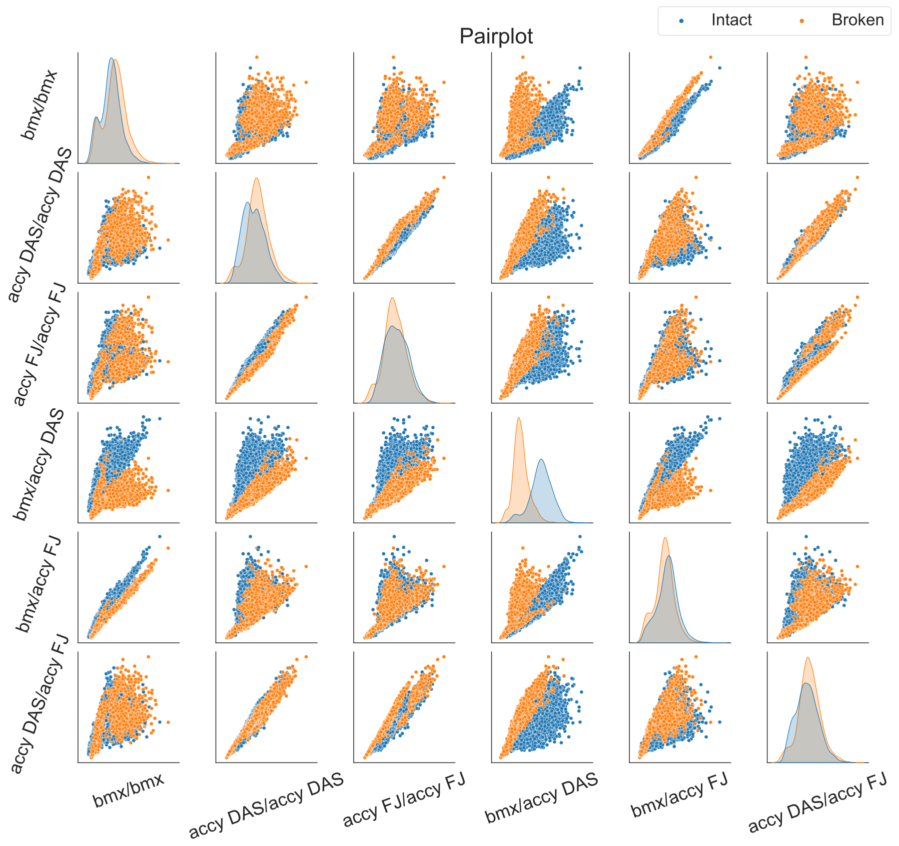

Figure 6.4: Pair plot of the data after using aforementioned covariance transform. For certain combinations the broken and intact cases separate quite well.

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 data. The algorithm starts by solving an eigenvalue problem for the covariance matrix $\boldsymbol{\Sigma}$. 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 $\boldsymbol{\Sigma}$. The eigenvectors form a basis for the data and the eigenvalues represent the distribution of the information of the source data.

Algorithm 6.1 Principal Component Analysis

```
1: function $\boldsymbol{P}$ = PCA($\boldsymbol{X}$):                          ▷ $\boldsymbol{X}$ - input, $\boldsymbol{P}$ - output
2:     $\boldsymbol{X} \longrightarrow \frac{\boldsymbol{X}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$    ▷ Normalize the data: $\boldsymbol{\mu}$ - mean, $\boldsymbol{\sigma}$ - standard deviation
3:     $\boldsymbol{\Sigma} = \frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X}$                 ▷ Calculate the covariance matrix
4:     $\boldsymbol{V}^T\boldsymbol{\Sigma}\boldsymbol{V} = \boldsymbol{D}$          ▷ Compute eigenvectors and eigenvalues of $\boldsymbol{\Sigma}$
5:     $\boldsymbol{W} = \left[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \ldots, \boldsymbol{w}_d\right]$  ▷ Transformation matrix consisting on the
       first $d$ eigenvectors of $\boldsymbol{V}$ arranged in order of decreasing eigenvalues
6:     $\boldsymbol{P} = \boldsymbol{X}\boldsymbol{W}$                      ▷ Project the data onto the new basis
7: end function
```

The goal is to choose a small enough subset of d eigenvectors corresponding to the d largest eigenvalues of $\boldsymbol{\Sigma}$. 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.

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.

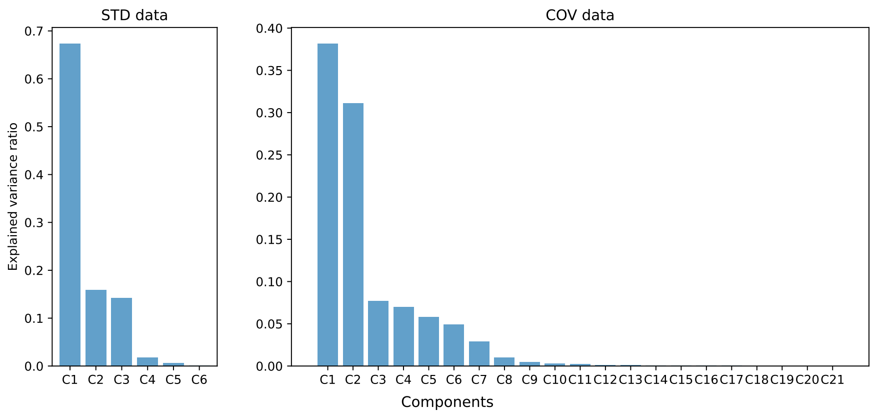

Figure 6.5: Ratio each component explains.

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

$$\begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix} = \begin{bmatrix}x^\top x & x^\top \mathbb{1}\\ \mathbb{1}^\top x & \mathbb{1}^\top \mathbb{1}\end{bmatrix}^{-1} \begin{bmatrix}x^\top\\ \mathbb{1}^\top\end{bmatrix} y, \qquad (6.3.1)$$

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.

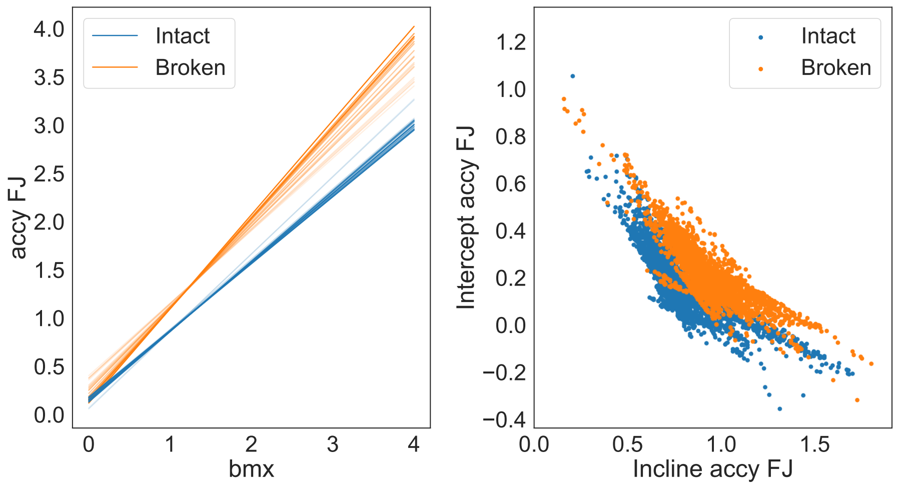

Figure 6.6: Left: Visualization of the lines capturing the relation between the standard deviation of accelerations in the flex-joint and wellhead bending moments using linear regression. The lines are meant to be displayed on a vessel's monitor and gradually fade over time highlighting the most recent 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

$$\begin{aligned}\beta_0 &= \mu_y - \frac{\mu_x \operatorname{Cov}(x,y)}{\operatorname{Var}(x)},\\ \beta_1 &= \frac{\operatorname{Cov}(x,y)}{\operatorname{Var}(x)}.\end{aligned}$$

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 *Target* = {*Broken, Intact*} by modelling the class probability $P = \Pr\left(\mathit{Target} = \mathit{Intact} \mid \boldsymbol{P}\right)$ as

$$\log \frac{P}{1-P} = \beta_0 + \beta^\top \boldsymbol{P},$$

with an intercept β_0 and a parameter vector β. The class probability is defined as

$$P = \frac{\exp\left(\beta_0 + \beta^\top \boldsymbol{P}\right)}{1 + \exp\left(\beta_0 + \beta^\top \boldsymbol{P}\right)}.$$

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 6.1, 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.

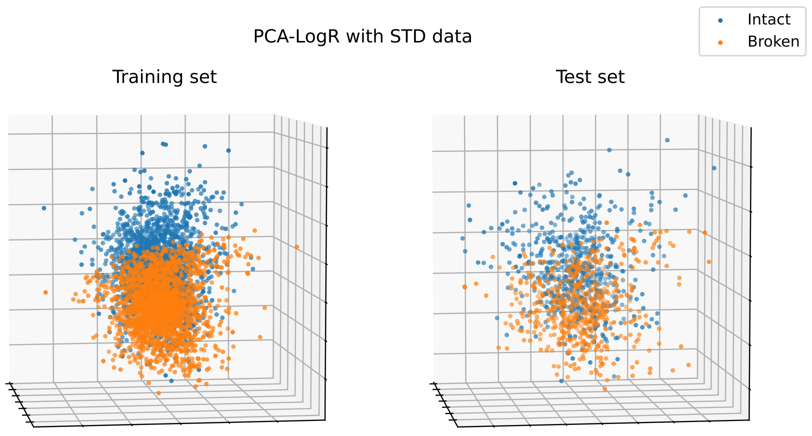

Figure 6.7: Classification of the STD data from the *Noise 1* data set with 3 principal components.

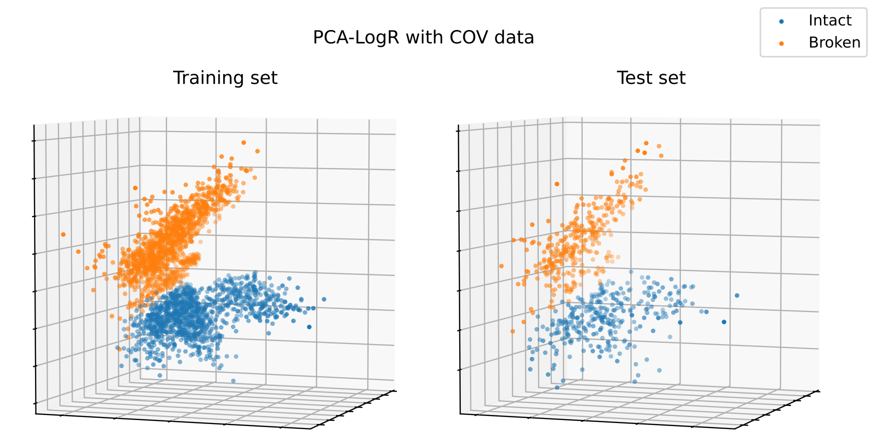

Figure 6.8: Classification of the COV data from the *Noise 1* data set with 4 principal components. The 3D visualization is made with 3 components.

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.

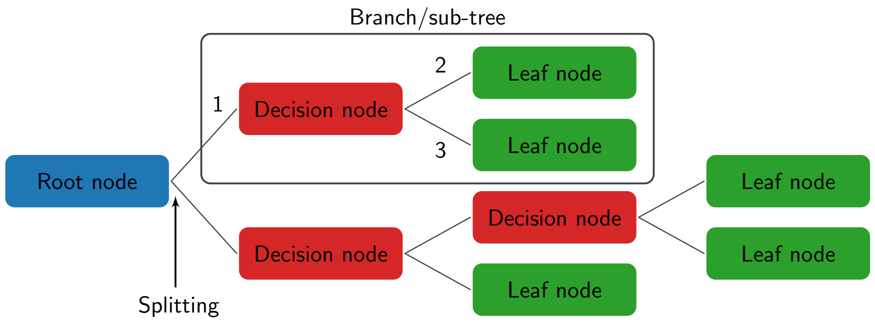

Figure 6.9: Example of a horizontal decision tree with depth 3. Node 1 is the parent node of nodes 2 and 3.

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*, [22, 30, 35].

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:

$$E_d = - \sum_{\substack{k=1 \\ p_{d,k} \neq 0}}^{l} p_{d,k} \log_2 \left(p_{d,k}\right), \tag{6.5.3}$$

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 [29]. The choice varies significantly on the particular circumstances and the data set. One advantage of the GI to the entropy approach

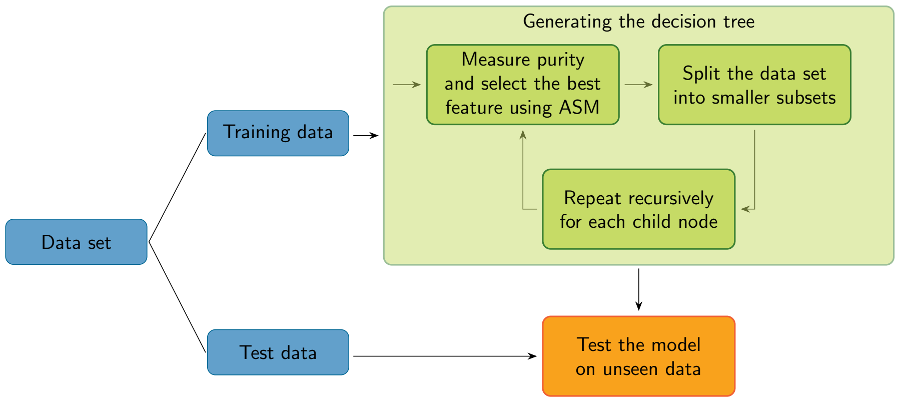
Generating the decision tree
Measure purity and select the best feature using ASM
Split the data set into smaller subsets
Repeat recursively for each child node
Training data
Data set
Test data
Test the model on unseen data

Figure 6.10: Decision tree algorithm illustrated as in [15].

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.

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 [19] which uses `gini` as splitting criterion and considers a binary split for each attribute. When `entropy` is chosen as splitting criterion, the ID3 algorithm [28] 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.

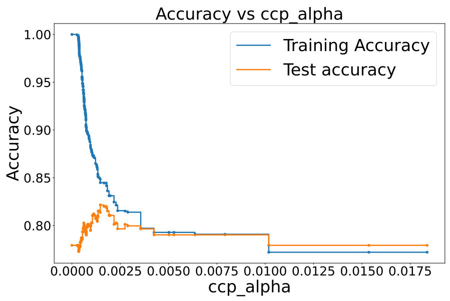

Figure 6.11: The effect of post-pruning in the reduction of overfitting. Scenario: Noise 50, COV-PCA(4), Gini (bottom-right block of Table 6.3.)

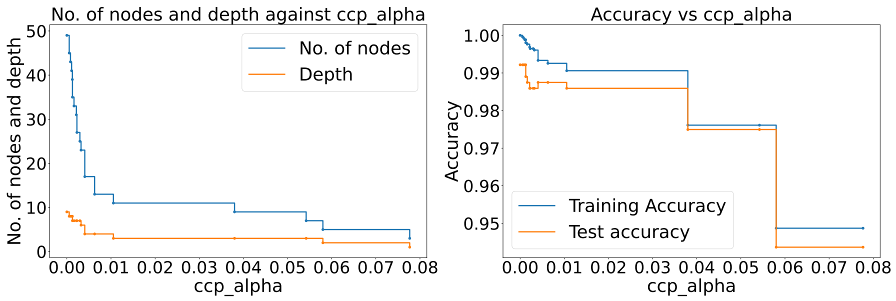

Figure 6.12: The effect of `ccp_alpha` on the structure and the accuracy of the tree. Scenario: Noise 1, COV-PCA(4), Entropy (marked in bold in Table 6.3.)

			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.

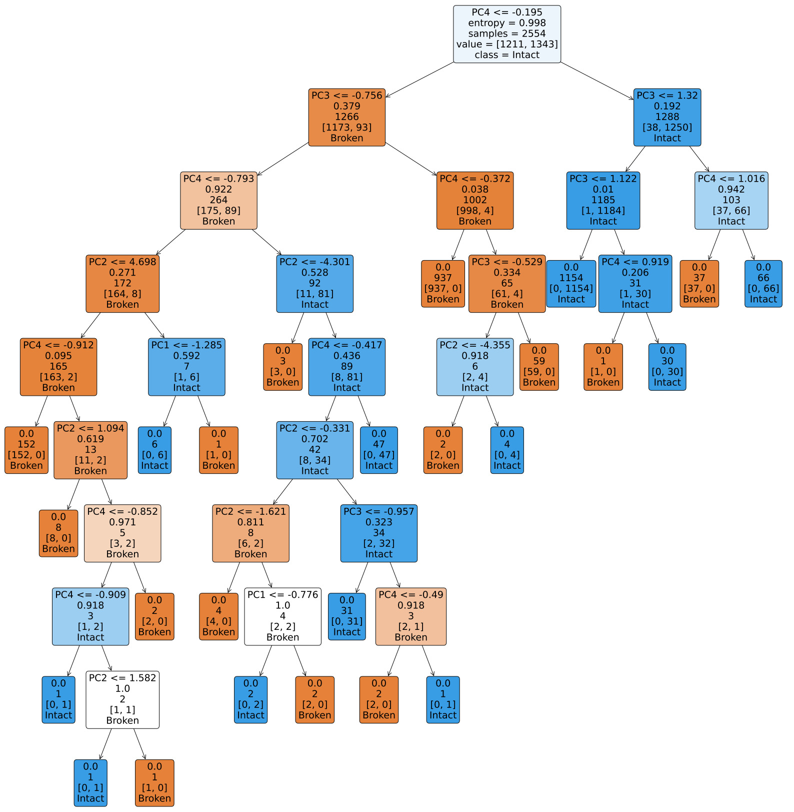
Figure 6.13: DT generated with entropy as splitting criterion on the data set consisting of the first four PCs of the COV data. Blue and orange are used for intact and broken, respectively. A light colour indicates a high entropy, an intense colour a low entropy.

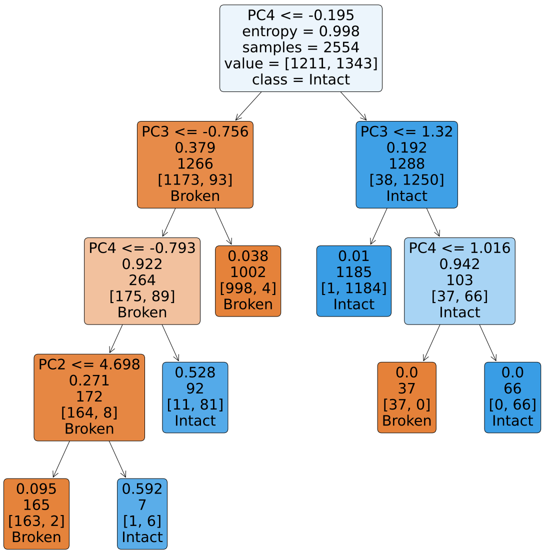
Figure 6.14: The same DT as in Figure 6.13 post pruned with `ccp_alpha` = 0.01.

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 [5], 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

$$D(x) = \sum_{j=0}^{l} y_j \alpha_j K\left(x_j, x\right) + b. \tag{6.6.2}$$

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 [5], the authors demonstrate that training the ML method involves solving a convex quadratic program. The soft margin was later introduced in [8], using l_2-penalization of mislabelled data points, thereby allowing for a feasible solution in the case of overlapping classes. Our model is trained by solving the quadratic program that follows,

$$\begin{array}{llcll}
 & \text{Primal} & & & \text{Dual} \\
\min\limits_{\alpha,\xi,b} & \frac{1}{2}\omega^2 + C\xi^\top \mathbb{1} & & \min\limits_{\alpha} & \frac{1}{2}\alpha^\top H\alpha - \alpha^\top \mathbb{1} \\
\text{s.t.} & y_i\left(\omega^\top \phi\left(x_i\right) + b\right) > 1 - \xi_i & & \text{s.t.} & \alpha^\top Y = 0 \\
 & \xi_i \geq 0 & & & 0 \leq \alpha \leq C\mathbb{1}, \\
 & \text{for all } i & & &
\end{array}$$

and differs slightly from the original method in [5] 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 $H_{ij} = y_i y_j K\left(x_i, x_j\right)$. 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.

Data	Noise 1			Noise 10			Noise 50		
trans.	Linear	RBF		Linear	RBF		Linear	RBF	
(#PCs)	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.

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, indi-

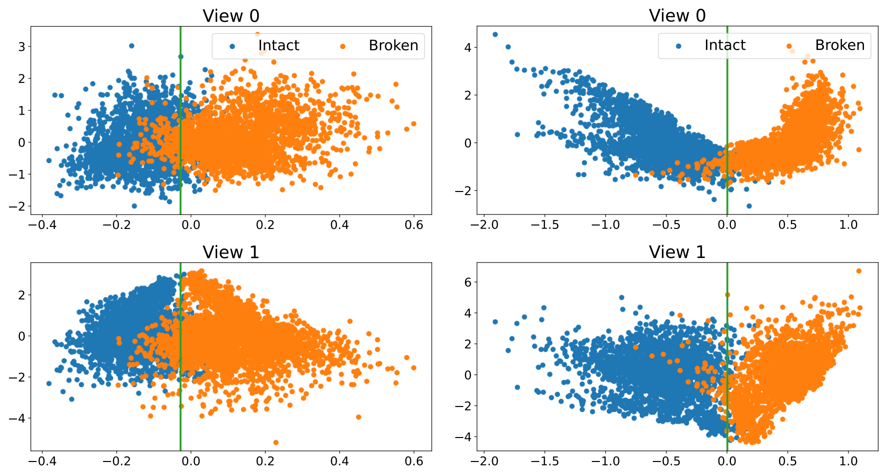

Figure 6.15: Figure showing linear SVMs performance on dataset with STD transform (left column) or COV transform and 3 PCs (right column). Both are created from a subset of the data set containing only one physical direction.

cating 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:

$$F_\theta\left(x_0\right) = \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_l \circ \ldots \phi_1\left(x_0\right), \tag{6.7.1}$$

with $x_0 \in \mathbb{R}^{n_0}$ a given input data, L the number of layers in the network, which determines its depth, and $\phi_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$, $\phi_l\left(x_{l-1}\right) := \sigma\left(W^l x_{l-1} + b^l\right)$ 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 component-wise, 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 $\{1-s, 2-s, \dots, s-2, s-1\}$, 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 [11].

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

$$\phi_{cn} : \mathbb{R}^{n_{l-1} \times m_{l-1}} \to \mathbb{R}^{n_l \times m_l}, \qquad \phi_{cn}(x_{l-1}) = P\Big(\sigma\Big(C\big(x_{l-1}\big)\Big)\Big),$$

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 [6]. 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.

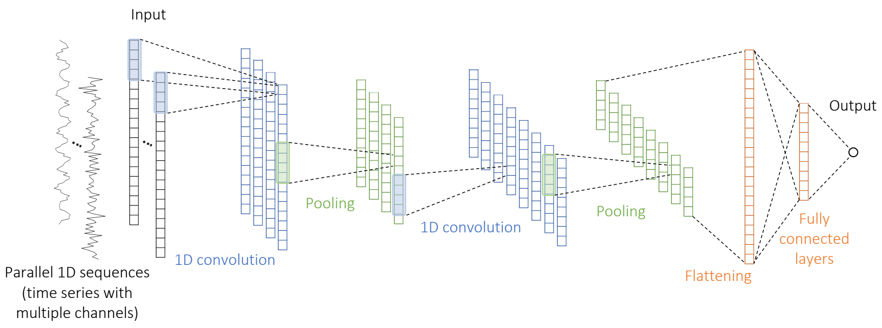

Figure 6.16: A typical one-dimensional CNN architecture.

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` [25].

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 [16], 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`.

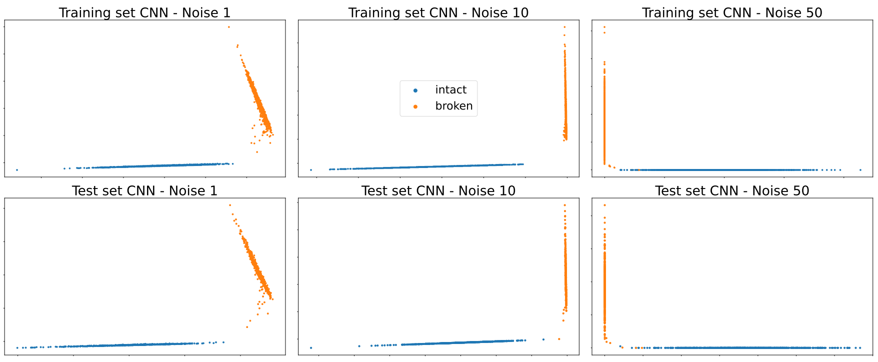

Figure 6.17: The figures illustrate the transformation of the input data by the CNN in both the training and test sets, under the three different noise scenarios. Prior to the output layer, which predicts the class, each individual time series is converted into a two-dimensional vector and can be visually 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.

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.

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

		Actual values	
		Broken	Intact
Predicted values	Broken	TP true positives - number of correctly classified broken wells	FP false positives - number of wrongly classified broken wells
	Intact	FN false negatives - number of wrongly classified intact wells	TN true negatives - number of correctly classified intact wells

Figure 6.18: Confusion matrix used to evaluate the performance of the classification techniques.

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 preprocessing 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.

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

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.

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 [31], autoencoders [2], CNN with Long Short Term Memory algorithms [3] or isolation forests [20], which have shown good results for anomaly detection.

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 [24]. 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).

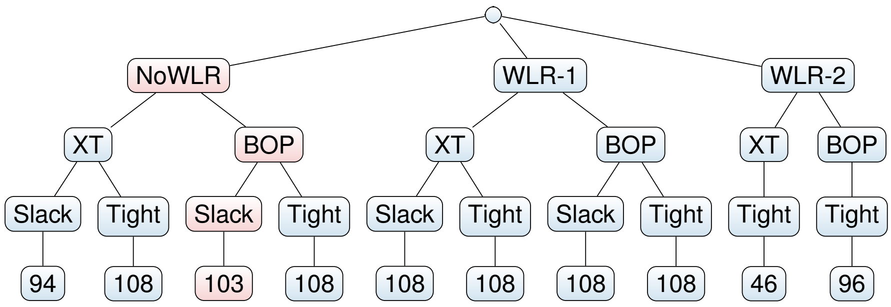

Figure 6.19: Number of analyses for fixed configurations. In red is the combination of configurations that we analyze in this work.

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.

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.

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.

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.

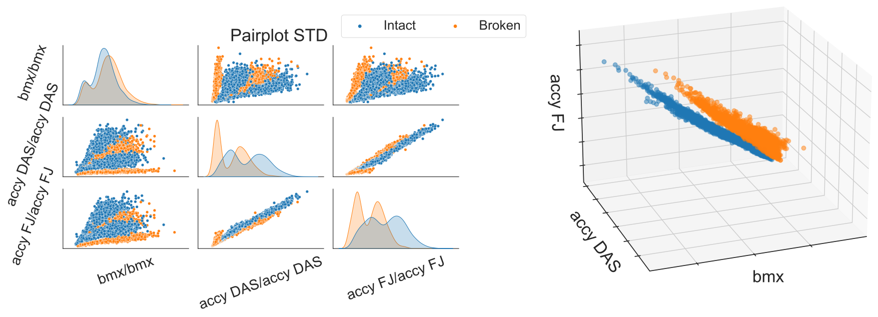

Figure 6.20: To the left a pair plot of the data after using aforementioned standard deviation transform on wells with a tight wellhead housing. For certain combinations the broken and intact cases separate quite well. To the right a 3D plot showing the spread of the data.

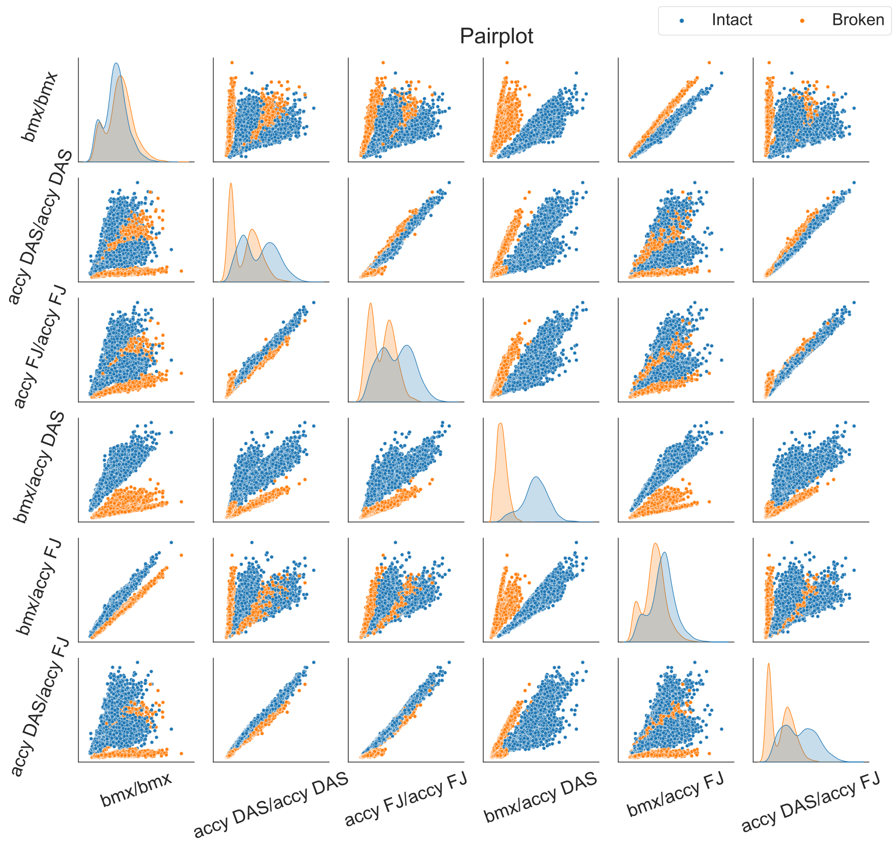

Figure 6.21: Pair plot of the data after using aforementioned covariance transform on wells with a tight wellhead housing. For certain combinations the broken and intact cases separate quite well.

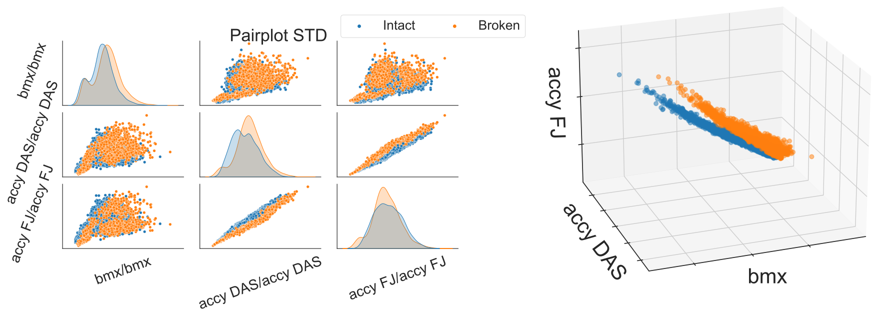

Figure 6.22: To the left a pair plot of the data after using aforementioned standard deviation transform on wells with a slack wellhead housing. For certain combinations the broken and intact cases separate quite well. To the right a 3D plot showing the spread of the data.

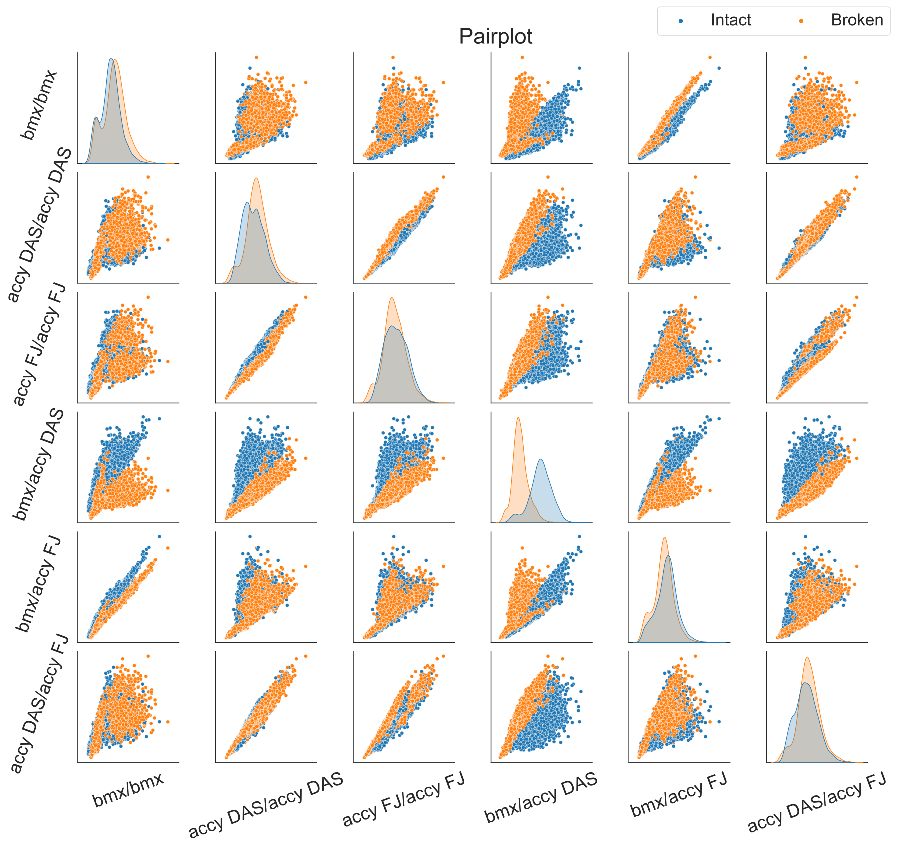

Figure 6.23: Pair plot of the data after using aforementioned covariance transform on wells with a slack wellhead housing. For certain combinations the broken and intact cases separate quite well.

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 $\alpha = 0.003$.

6.C Supplementary material for the reproducibility of the experiments: Convolutional Neural Networks

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`.

```
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.

Acknowledgements This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 860124. This publication reflects only the author's view and the Research Executive Agency is not responsible for any use that may be made of the information it contains.

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.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[2] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, Machine learning for data science handbook: data mining and knowledge discovery handbook (2023), 353–374.

[3] 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** (2023), no. 5, 28–44.

[4] J. Berkson, *Application of the logistic function to bio-assay*, Journal of the American Statistical Association **39** (1944), no. 227, 357–365.

[5] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the Fifth Annual Workshop on Computational Learning Theory (New York, NY, USA), COLT '92, Association for Computing Machinery, 1992, p. 144–152.

[6] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*, 2021, arXiv:2104.13478.

[7] 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** (2020), 189–215.

[8] C. Cortes and V. Vapnik, *Support-vector networks*, Machine Learning **20** (1995), no. 3, 273–297.

[9] 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** (2019), no. 4, 917–963.

[10] F. E. Harrell, *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, vol. 608, Springer, 2001.

[11] C. F. Higham and D. J. Higham, *Deep learning: An introduction for applied mathematicians*, Siam review **61** (2019), no. 4, 860–891.

[12] H. Hotelling, *Analysis of a complex of statistical variables into principal components*, J. Educ. Psychol. 24 (1933), 417–441.

[13] J. Cadima I. T. Jolliffe, *Principal component analysis: a review and recent developments*, Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences **374** (2016), no. 2065, 20150202.

[14] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, vol. 398, John Wiley & Sons, 2013.

[15] H. Kim, *Supervised learning*, pp. 87–182, Springer International Publishing, Cham, 2022.

[16] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, International Conference on Learning Representations (ICLR) (San Diega, CA, USA), 2015.

[17] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature **521** (2015), no. 7553, 436–444.

[18] D. Lee, S. Malacarne, and E. Aune, *Vector quantized time series generation with a bidirectional prior model*, Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, eds.), Proceedings of Machine Learning Research, vol. 206, PMLR, 4 2023, pp. 7665–7693.

[19] R. J. Lewis, *An introduction to classification and regression tree (cart) analysis*, Annual meeting of the society for academic emergency medicine in San Francisco, California, vol. 14, Citeseer, 2000.

[20] F. T. Liu, K. M. Ting, and Z.-H. Zhou, *Isolation forest*, 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413–422.

[21] S. Menard, *Logistic regression: From introductory to advanced concepts and applications*, Sage, 2010.

[22] F. Mola and R. Siciliano, *A fast splitting procedure for classification trees*, Statistics and Computing **7** (1997), 209–216.

[23] J. N. Morgan and J. A. Sonquist, *Problems in the analysis of survey data, and a proposal*, Journal of the American statistical association **58** (1963), no. 302, 415–434.

[24] Orcina Ltd, *Orcaflex*.

[25] 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, vol. 32, Curran Associates, Inc., 2019.

[26] K. Pearson, *On lines and planes of closest fit to systems of points in space*, Phil. Mag. 2 (1901), 559–572.

[27] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman, *Decision trees: an overview and their use in medicine*, Journal of medical systems **26** (2002), 445–463.

[28] J. R. Quinlan, *Induction of decision trees*, Machine learning **1** (1986), 81–106.

[29] L. E. Raileanu and K. Stoffel, *Theoretical comparison between the gini index and information gain criteria*, Annals of Mathematics and Artificial Intelligence **41** (2004), 77–93.

[30] L. Rokach and O. Maimon, *Top-down induction of decision trees classifiers-a survey*, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **35** (2005), no. 4, 476–487.

[31] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, *Support vector method for novelty detection*, Proceedings of the 12th International Conference on Neural Information Processing Systems (Cambridge, MA, USA), NIPS'99, MIT Press, 1999, p. 582–588.

[32] CTi Sensors, *TILT – 57A DYNAMIC INCLINOMETER, Three-Axis Accelerometer, Three-Axis Gyroscope*, 2024, `https://ctisensors.com/products/tilt-5x-dynamic-inclinometer/`.

[33] R. H. Shumway and D. S. Stoffer, *Time series analysis and its applications*, Springer Cham, 2017.

[34] Y. Y. Song and L. Ying, *Decision tree methods: applications for classification and prediction*, Shanghai archives of psychiatry **27** (2015), no. 2, 130.

[35] S. 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** (2020), no. 2, 612–619.

[36] A. Venkatasubramaniam, J. Wolfson, N. Mitchell, T. Barnes, M. Jaka, and S. French, *Decision trees in epidemiological research*, Emerging themes in epidemiology **14** (2017), 1–12.

[37] Det Norske Veritas, *Recommended practice*, Tech. Report DNV-RP-E104, `https://www.dnv.com/oilgas/download/dnv-rp-e104-wellhead-fatigue-analysis.html`, 01 2019, DNV AS.

ISBN 978-82-326-8098-6 (printed ver.)
ISBN 978-82-326-8097-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology