Physics-Informed Machine Learning for the Earth Sciences: Applications to Glaciology and Paleomagnetism

by

Facundo Fabián Sapienza

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Statistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Fernando Pérez, Co-chair
Professor Jonathan Taylor, Co-chair
Assistant Professor Ryan Giordano
Assistant Teaching Professor Alexander Strang

Spring 2024

Physics-Informed Machine Learning for the Earth Sciences:
Applications to Glaciology and Paleomagnetism

Abstract

Physics-Informed Machine Learning for the Earth Sciences:
Applications to Glaciology and Paleomagnetism

by

Facundo Fabián Sapienza

Doctor of Philosophy in Statistics

University of California, Berkeley

Associate Professor Fernando Pérez, Co-chair

Professor Jonathan Taylor, Co-chair

This dissertation studies the application of machine learning in the fields of Glaciology and Paleomagnetism. In the past few years, there have been significant advances in introducing physical constraints in the form of inductive biases in data-driven approaches coming from statistics and machine learning. This gave rise to the field of physics-informed machine learning, which we will introduce in Chapter 1. Chapters 3 and 4 will cover the application of neural differential equations for ice flow modelling, showcasing how the differentiable programming techniques introduced in Chapter 2 have been successfully applied for the inversion and calibration of the internal ice viscosity of mountain glaciers with different climates. This led to the development of `ODINN.jl`, a multilanguage Julia-Python package for the modelling of global glacier-climate interactions. We will finalize our discussion in Chapter 5 with the quantification of errors involved in paleomagnetic sampling and further applications of non-parametric regression based on neural differential equations.

Many years later as he faced the firing squad,
Colonel Aureliano Buendía was to remember that distant afternoon
when his father took him to discover ice.

José Arcadio Buendía ventured a murmur:
"It's the largest diamond in the world."
"No," the gypsy countered. "It's ice."

Five reales more to touch it," he said.
José Arcadio Buendía paid them and put his hand on the ice
and held it there for several minutes as his heart
filled with fear and jubilation at the contact with mystery.

"This is the great invention of our time."

100 Años de Soledad, Gabriel García Márquez

Malchin Peak, in the triple border between Mongolia, Russia, and China. Picture taken a few months before embarking on the doctoral endeavour.

Contents

List of Figures

List of Tables

Introduction

Geoscientific models are facing increasing challenges to take advantage of growing datasets coming from remote sensing. Physics-informed machine learning, aided by differentiable programming, provides a new scientific modelling paradigm enabling both complex functional inversions to potentially discover new physical laws and data assimilation from heterogeneous and sparse observations.

Machine learning methods have opened new avenues for extending traditional physical modeling approaches with rich and complex datasets, offering advances in both computational efficiency and predictive power. Nonetheless, the lack of interpretability of some of these methods, including artificial neural networks, has been a frequent subject of concern when modelling physical systems (Zdeborová 2020). This *black box* effect is particularly limiting in scientific modelling, where the discovery and interpretation of physical processes and mechanisms plays a central role to improve our understanding of the physical system under study. As a consequence, a new breed of machine learning models has appeared in the last few years, attempting to add physical constraints and interpretability to learning algorithms (Chen et al. 2018; Rackauckas et al. 2020; Raissi et al. 2019).

In this thesis, we will focus our attention to two fields in geophysics, namely glaciology and paleomagnetism. Further applications to planetary sciences are included in Publications.

Glaciology

Large parts of this thesis concern the study of the cryosphere, that is, the study of ice on Earth. More specifically, we will focus our attention on glaciology and the study of how ice masses move in mountain glaciers and the large ice sheets and shelves in Antarctica and Greenland, and how that flow is affected by the climate and other external conditions.

Glacier modeling in the 21st century is an important piece of the complex puzzle to understand the implications that climate change will have on Earth. One of the long-standing questions in glaciology and sea-level predictions is how internal ice deformation and ice-bedrock sliding laws respond to different stresses and physical variables, such as air and ice temperature and water in the subglacial drainage system.

Remote sensing observations have sparked a revolution in scientific computing and modeling within Earth sciences, and glaciology is no exception (Hugonnet et al. 2020; Millan et al. 2022). Historically, the field of glaciology has been based on the generalization of properties derived from the calibration and modeling of individual glaciers based on in-situ measurements. However, in the past decade there has been an increase in remote-sensing datasets providing ice surface velocities of mountain glaciers (a total of approximately 274, 000 mountain glaciers distributed around the globe (RGI 7.0 Consortium 2023)), ice sheets, and ice shelves (Farinotti et al. 2019; Millan et al. 2022; Morlighem et al. 2020), together with ice-penetrating radar thickness observations (Schroeder 2022).

This revolution is assisted by modelling frameworks based on machine learning (Jouvet et al. 2021; Rasp et al. 2018), computational scientific infrastructure, e.g. Jupyter and Pangeo (Arendt et al. 2018; Kluyver et al. 2016), and modern programming languages like Python and Julia. In glaciology, classification methods have been more popular than regression methods (Baumhoer et al. 2019; Mohajerani et al. 2019). Nonetheless, progress has been made with surrogate models for ice flow modelling (Jouvet et al. 2021; Riel et al. 2021), subglacial processes (Brinkerhoff et al. 2020), glacier mass balance modelling (Anilkumar et al. 2022; Bolibar et al. 2020a,b; Guidicelli et al. 2023) or super-resolution applications to downscale glacier ice thickness (Leong et al. 2020). In glaciology, and more specifically in ice flow modelling, it is still challenging to move from small-scale detailed observations and physical processes (for example, the viscous properties of ice derived with ice in the laboratory) to large-scale observations and parametrizations used to model ice flow at the scale of Antarctica or Greenland. When modelling glaciers globally, simple empirical models such as temperature-index models are used, due to their robustness to noise and the lack of observations needed to support more complex models. The same applies for ice flow dynamics, with flowline models based on the Shallow Ice Approximation (SIA, (Hutter 1983)) being widely used, as it provides a good approximation, particularly with noisy and coarse-resolution input data typical from large-scale models (Maussion et al. 2019; Zekollari et al. 2019). Moreover, it helps to reduce the computational cost of simulations when compared to higher-order models. Therefore, there is a broad need for new methods that enable a robust calibration and discovery of more sophisticated, nonlinear interpretable parametrizations, in geosciences in general, but also for both glacier mass balance and ice flow dynamics. These include the need to transition towards non-constant melt and accumulation factors for temperature-index mass balance models (Bolibar et al. 2022), or the need to find a robust relationship to calibrate ice creep and basal sliding for different regions, topographies and climates (Hock et al. 2023).

This motivated the development of `ODINN.jl`, a new modelling framework based on universal differential equations (UDEs) applied to glacier ice flow modelling. UDEs or neural differential equations are a particular type of algorithms that embed neural networks inside a differential equation (Rackauckas et al. 2020). In Chapter 4, we illustrate how UDEs, supported by differentiable programming in the Julia programming language, can be used to infer empirical laws present in datasets, even in the presence of noise. We did so by using a prescribed artificial law as a subpart of the partial differential equation used to model ice flow. We used a neural network to infer the functional dependency between the internal ice viscosity with respect to a climatic proxy for 17 different glaciers across the world. The presented functional inversion framework is robust to noise present in input observations, particularly on the surface mass balance, as shown in an experiment.

Paleomagnetism

Another domain in geophysics where statistical and machine learning modelling is gaining momentum is Paleomagnetism. Paleomagnetism is the study of the ancient geomagnetic field of the Earth based on the measurement of remanent magnetization present in rocks. These are used in paleogeography for the reconstruction of the past motion of tectonic plates (Tauxe et al. 2003).

Paleomagnetic data are typically assembled and reported in a hierarchical fashion (Tauxe 2010). First, individual paleomagnetic sample directions are grouped and averaged into sites. Sites are geological units for which the magnetic remanence direction of all associated samples is interpreted to have recorded the same spot reading of the geomagnetic field. Given that the timescale of remanence acquisition is typically quite short relative to the timescale of secular variation of Earth's magnetic field, individual sites do not average out this variability. A virtual geomagnetic pole (VGP) calculated from site-level data thus represents the position of the magnetic dipole axis at the time of magnetization acquisition. Traditionally, the subsequent step is to average site-level VGPs to yield a study-level *paleomagnetic pole* (or *paleopole*), which, in the framework of the time-averaged geocentric axial dipole hypothesis, is interpreted to correspond to Earth's spin axis. Finally, study-level paleopoles are conventionally aggregated to construct an apparent polar wander path (APWP) representing the apparent motion of the spin-axis that can be used to reconstruct plate tectonic motion (Besse et al. 2002; Kent et al. 2010; Torsvik et al. 2012).

Due to the importance of how paleomagnetic observations are collected and interpreted as paleopoles, there is a need to quantify the uncertainty in these estimates and further improve the field sampling strategies, a problem that has long been recognized in the field. In Chapter 5 we will introduce a quantitative analysis based on both numerical simulation and theoretical results that shows the different trade-offs between different sampling strategies and makes a step forward in introducing quantifiable metrics at the moment of estimating paleomagnetic pole position and dispersion in paleomagnetic studies. Although not covered in the same chapter, further progress has been made in the methods employed for APWP estimation used to described the relative motion of tectonic plates in the past. Commonly, APWPs are generated by means of moving average techniques, wherein a Fisher mean is computed from all the paleopoles whose age falls within a running window (Irving 1977). However, these methods fail at the moment of detecting change points in the dynamics of plate motion, which lead to the development of new methodologies (Gallo et al. 2022; Sapienza et al. 2024b). Furthermore, a Monte Carlo uncertainty propagation scheme that operates on site-level paleomagnetic data has been introduced to integrate multiple sources of noise in the statistical construction of APWPs without relying on parametric assumptions of the underlying data (Gallo et al. 2023).

The rest of the thesis is organized as follows. Chapter 1 consists on an essay introducing the framework we will use in this thesis for the statistical modelling of the physical sciences. Chapter 2 includes a review of differentiable programming tools for differential equation systems and a quick overview of different physics-informed machine learning methods. Chapter 3 introduces the fundamentals of ice flow modelling and the necessary mathematical framework to formulate ice flow equations used inside `ODINN.jl`. Chapter 4 will discuss the development of the modelling framework in `ODINN.jl`. Finally, Chapter 5 will move the focus to paleomagnetism and will show results on optimal sampling strategies.

Publications

Most results included in this thesis are part of the following articles in press:

- J. Bolibar, F. Sapienza, F. Maussion, R. Lguensat, B. Wouters, and F. Pérez (2023a). "Universal differential equations for glacier ice flow modelling". In: *Geoscientific Model Development* 16.22, pp. 6671–6687. DOI: 10.5194/gmd-16-6671-2023[1]

- F. Sapienza, L. C. Gallo, Y. Zhang, B. Vaes, M. Domeier, and N. L. Swanson-Hysell (2023a). "Quantitative Analysis of Paleomagnetic Sampling Strategies". In: *Journal of Geophysical Research: Solid Earth* 128.11, e2023JB027211. DOI: https://doi.org/10.1029/2023JB027211

and the following list of manuscripts in preparation:

- F. Sapienza, J. Bolibar, F. Schäfer, P. Heimbach, G. Hooker, F. Pérez, P. Persson, C. Rackauckas, V. Boussange, B. Groenke, and A. Pal (2024c). "Differentiable Programming for Differential Equations: A Review". In: *preparation*

- F. Sapienza et al. (2024b). "Fitting curves in the sphere using universal differential equations". In: *preparation*

These further include a list of software tools developed along with the previous publications:

- J. Bolibar and F. Sapienza (June 2023b). *ODINN-SciML/ODINN.jl: v0.2.0*. Version v0.2.0. DOI: 10.5281/zenodo.8033313

- F. Sapienza, L. C. Gallo, Y. Zhang, B. Vaes, M. Domeier, and N. Swanson-Hysell (2023b). *PolarWandering/PaleoSampling (Version 1.0.0)*. Comp. software. Version 1.0.0. DOI: https://doi.org/10.5281/zenodo.8347149

This is a list of publications result from the collaboration between statisticians and planetary scientists during the period of this dissertations. Although this research was conducted entirely as part of the doctoral work, these results are non included in this dissertation:

- F. Sapienza et al. (2024a). "An Analytical Model of Magnetic Field Draping in Induced Magnetospheres". In: *preparation*

[1] J. Bolibar and F. Sapienza contributed equally to this work.

- ▶ A. R. Azari, E. Abrahams, F. Sapienza, D. L. Mitchell, J. Biersteker, S. Xu, C. Bowers, F. Pérez, G. A. DiBraccio, Y. Dong, and S. Curry (2023). "Magnetic Field Draping in Induced Magnetospheres: Evidence From the MAVEN Mission to Mars". In: *Journal of Geophysical Research: Space Physics* 128.11, e2023JA031546. DOI: https://doi.org/10.1029/2023JA031546

- ▶ A. Azari, E. Abrahams, F. Sapienza, J. Halekas, J. Biersteker, D. Mitchell, F. Pérez, M. Marquette, M. Rutala, C. Bowers, et al. (2024). "A Virtual Solar Wind Monitor for Mars with Uncertainty Quantification using Gaussian Processes". In: *arXiv preprint arXiv:2402.01932*

Further collaborations that lead to peer-review publications during the period of the doctoral program include:

- ▶ E. Smucler, F. Sapienza, and A. Rotnitzky (2022). "Efficient adjustment sets in causal graphical models with hidden variables". In: *Biometrika* 109.1, pp. 49–65

- ▶ L. C. Gallo, M. Domeier, F. Sapienza, N. L. Swanson-Hysell, B. Vaes, Y. Zhang, M. Arnould, A. Eyster, D. Gürer, Á. Király, B. Robert, T. Rolf, G. Shephard, and A. van der Boon (2023). "Embracing Uncertainty to Resolve Polar Wander: A Case Study of Cenozoic North America". In: *Geophysical Research Letters* 50.11. DOI: 10.1029/2023gl103436

- ▶ F. Chazal, L. Ferraris, P. Groisman, M. Jonckheere, F. Pascal, and F. Sapienza (2023). "Choosing the parameter of the Fermat distance: navigating geometry and noise". In: *arXiv preprint arXiv:2311.18663*

- ▶ F. Cerisola, F. Sapienza, and A. J. Roncaglia (2022). "Heat engines with single-shot deterministic work extraction". In: *Physical Review E* 106.3, p. 034135

- ▶ L. C. Gallo, F. Sapienza, and M. Domeier (2022). "An optimization method for paleomagnetic Euler pole analysis". In: *Computers & Geosciences* 166, p. 105150

- ▶ L. C. Gallo et al. (2024). "On the feasibility of paleomagnetic Euler pole analysis". In: *preparation*

Acknowledgments

I would like to start by acknowledging Fernando Pérez and Jonathan Taylor for being my advisors during these last five years. You are both wonderful and incredible mentors. You each deserved at least a paragraph, so here we go.

Muchas gracias Fernando por todas tus enseñanzas y por mostrarme una nueva manera de hacer ciencia e investigación. Viendo cinco años atras, todavia no entiendo muy bien como llegue hasta aca, haciendo lo que hacemos, pero sé con certeza que fue por haber confiado y creido en tu vision. Las enseñanzas y las conversacions de los últimos años me las quedo para siempre. Más que adiós, el fin de mi doctorado es un *hasta la próxima, Señor Director*.

Thank you Jonathan for taking me as your student even when we are in opposite sides of the Bay. I have learned a lot from you in the last five years and I am happy to know that I will be able to continue working together and interacting with you in my next academic stage. Looking forward for more hours-long conversations in your office! Thank you for being my statistician of reference.

During my doctoral studies, I was very fortunate to work closely with amazing postdocs: Abigail Azari, Jordi Bolibar, and Leandro Gallo. If Fernando and Jonathan were my academic parents during the last years, then you are my academic big brothers and sisters. You also each deserved a separate paragraph.

Thank you Abby for taking my research to the stars (sorry, I needed to make the joke). I enjoyed working with you and special thank you for your commitment to give me advice in multiple occasions along the course of the years.

Gracias Jordi por haberme incluido en la familia ODINN desde sus inicios. Es un placer trabajar con vos, y aprendo de vos cada día más. Sos una referencia para mi y estoy seguro de que nuestros caminos van a seguir juntos por un buen tiempo. Gracias por la amistad y por ser un mentor. Por mucha más ciencia juntos, más conciertos, más black metal, más cervezas, y por supuesto: *¡Visca ODINN!*.

Gracias Leo por haberme reincertado al mundo del Paleomagnetismo. Es un placer trabajar con vos y espero que esta sea una relación que se mantenga hasta nuestra jubilación, en alguna quinta de Buenos Aires. Sos una referencia y aprendí muchas cosas de vos a lo largo de los años. Muchas gracias, amigo, colega, y referencia (en otras palabras, titán).

Thank you to the members of my qualifying exam, dissertation committee, and the many professors and mentors I was able to learn from in the past five years. Thank you Alexander Strang, Giles Hooker, Kurt Cuffey, Mathew Domeier, Nicholas Swanson-Hysell, Per-Olof Persson, Ryan Giordano.

To all the Berkeley crew, including professors, students, and staff. Special thank you to the students and friends in my cohort who helped me in multiple instances in the past (Adam, Alice, Corrine, Yassine) and the wonderful staff of Berkeley Statistics, special gratitude to La Shana Porlaris, Ryan Lovett, and Tanisha Robinson.

Thank you to all the people I was able to collaborate. Thank you Lindsey Heagy for helping me on my first steps in the group (and helping on on my first PR back in the days). Special thank you to all the members of the Cryo group: Ben Hills, Ellianna Abrahams, James Butler, Matthew Siegfried, Shane Grigsby, Tasha Snow, Whyjay Zheng. Thank you to the paleomag crew in Berkeley, including Nick and Yiming.

Muchas gracias a mi familia por haberme bancado durante todos estos años y desde la distancia haber acompañado este sueño. No estaria aca de no ser por la educación, enseñanzas y cariño que recibí desde chico. Gracias Mamá, Papá, Cami, Lela, Titi, Tio Nano, Bauti, Tio Gus, y Tia Adriana.

A los amigos de fierro que siguen estando a lo largo de todos estos años. Gracias Lulú, la casa está en construcción. Gracias Agus, Dani, Lauta, Lucho, Manu, Pili, Teclo.

Thank you to the members of the Cedar Scholars Association for their camaraderie during the last three wonderful years. Thank you Adam, Adele, Ellen, Frank, Georgia, Guillaume, Kris, Paola, Sophie, Willie.

Thank you to all the friends I made during my time at Berkeley. Thank you to all the Argentinian crew. Thank you Meli for being my friend since my first day at Berkeley and helping me in this new journey.

I am a product of high-quality, public, and free education. Thank you to the University of Buenos Aires and the Argentinian public educational system. I wouldn't be here without the tons of professors and instructors at each stage of education I was immerse in since I was 3 years old. Special thanks to the people responsible for why I am here right now: Augusto Roncaglia, Carlos Vazquez, Leonardo Boechi, Matthieu Jonckheere, Pablo Groisman.

Finally, thank you so much to the people who offered their help and support during the last round of this journey. Thank you Adam, James, Jimmy, and Margaret.

Chapter 1

Statistical modelling in the physical sciences

In this first chapter I am going to take on the challenge of answering the following question that has driven my decision to pursue a doctorate in Statistics:

> **Question No. 1.** *What does statistical modelling and machine learning have to offer the physical and Earth sciences?*

Furthermore, as scientists we have the job of finding links between our present scientific goals and new methodologies to address them. Nature does not care about the artificial boundaries we impose between disciplines, let alone the distinction between departments in the university. Understanding the world around us and the interconnectivity of its parts is a joint scientific effort, and the search for knowledge and understanding is a universal human effort. In the same spirit, the second question that I am interested in addressing here is the following:

> **Question No. 2.** *What does Earth Science modelling have to offer Statistics?*

Many of the modelling and data-related challenges found in the physical sciences today are of a statistical nature, included but not limited to geospatial model validation, introduction of priors in form of mechanistic equations, and uncertainty quantification.

Today the need to understand how the Earth works and how it will respond to climate change is essential to anticipate what our future on Earth will be like and what measures we can take in time to alleviate its effects. To be successful, as a community we need to start thinking about a collective effort where scientists and humans with different abilities come together and embrace the recent advances in computational sciences, data science, and statistics. It is my own belief, the belief of many of my colleagues, and many members of the scientific community that the next breakthrough in physics and Earth sciences will come from the combination of data science, domain knowledge, and data and software engineering. Here we stand.

1.1 Why now?

In this first section, I will explain what I believe are some of the reasons why there is an increase in efforts to combine machine learning with the physical sciences. While most of this discussion applies to different domains, from fundamental physics to biology, our discussion will focus on Earth sciences and geophysics, and even more specifically to glaciology, with some remarks to paleomagnetism and planetary sciences.

Increase of scientific data sets. The fuel of any statistical, data science, and machine learning algorithm is data. In fundamental fields in physics, traditional computational methods are no longer able to handle the massive volumes of data being collected every day. Examples of this include gravitational-wave detection (Cuoco et al. 2020) and high-energy physics (Maguire et al. 2017). In the case of the Earth sciences, many datasets consist of re-analyses of remote sensing products with global coverage and fine spatial and/or temporal resolution, allowing to sample the dynamic range of our planet. In glaciology, the last few years had witnessed an increase in remote-sensing datasets providing ice surface velocities of mountain glaciers (Millan et al. 2022), ice sheets, and ice shelves (Farinotti et al. 2019; Morlighem et al. 2020); ice surface altimetry (Abdalati et al. 2010; Magruder et al. 2021); and ice-penetrating radar thickness observations (Schroeder 2022).

How do we deliver the data? More than just data accumulation, recent years have seen an increase in standardization and the creation of data science-ready datasets that are openly avaliable to the scientific community (Peckham 2014). This includes the creation of analysis-ready data, cloud-optimized (ARCO) formats (Abernathey et al. 2021). Journals, publishers, and agencies like NASA are now moving forward in the internet-based democratization of data, supporting the open access to data and software (National Academies of Sciences and Division on Engineering and Physical Sciences et al. 2018). An outstanding example of this is icepyx, a open source Python library design to access data from the ICESat-2 laser altimeter satellite mission (The icepyx Developers 2023). Further examples of data-science ready datasets in other fields include ClimateSet in climate sciences (Kaltenborn et al. 2023), the materials project (Jain et al. 2013), and the open catalyst project in material design (Chanussot et al. 2021).

Do we need more data or better models? In areas such as computer vision and natural language processing, the importance of massive datasets over sophisticated algorithms has been highlighted in the literature as the main factor driving the success of machine learning (Halevy et al. 2009; Weyand et al. 2016). This point has been raised in the essay *The Unreasonable Effectiveness of Data* (Halevy et al. 2009) which challenges the traditional success of the physical science by relying on mathematical principles, a point made by Nobel laureate in physics Eugene Wigner in *Unreasonable Effectiveness of Mathematics in the Natural Sciences* (Wigner 1960). However, data in the Earth sciences are sparse, noisy, and even if they are large, they are not massive enough to be trained with physics-free models (Karpatne et al. 2017b). This naturally leads our discussion towards the advances in designing new algorithms that combine the physics and data-based approaches.

Search for new models of Nature. As our understanding of the fundamental behaviour of the natural world increases, it becomes more difficult to resolve detailed aspects of scientific theories by simple observation and speculation. For new and emerging theories, this works very well and it has been the successful path that disciplines like physics have followed for centuries. However, as the complexity of existing theories increases and scientists start asking more complicated questions, the observation-assimilation approach becomes more difficult. Complex physical systems are governed by typically unknown causal mechanisms (Ebert-Uphoff et al. 2014; Runge et al. 2019). Some of them are chaotic (weather), others higher-non linear (ice flow), and in general they are affected by mechanisms we do not fully comprehend (eg, friction laws between rock and ice in the bed of a glacier).

Another obstacle in Earth science modelling is that large-scale systems are dominated by processes that typically operate at different spatiotemporal resolutions. The traditional example for this is climate sciences, where convection, storms, and cloud processes operate at different scales (Schneider et al. 2017). In glaciology, the composition of ice crystals in the scales of 100-1000 nanometers have an impact in the shearing properties of ice use for large-scale modelling of ice sheets and glaciers (Cuffey et al. 2010). Furthermore, there is a large discrepancy between laboratory and site measurements. An example of this is the calibration of rheological laws regarding the deformation of ice under different stresses, a phenomena described by Glen's law that we will discuss in Section 3.1.2. Laboratory measurements are performed on ice blocks no larger than one meter, even when derived results will later be used to model large-scale ice sheets like Antarctica and Greenland.

Both the evident complexity of the natural world and the increasing amount of available data to validate our models set limits on how much more we can learn about Nature by simple observation and speculation. Traditional physical theories lack algorithmic complexity (Roberts 2021). It is in this context that we have been seen new research frameworks with adjusted models that represent a compromise between the data-driven and physics-based traditions (Azari et al. 2020; Karniadakis et al. 2021; Karpatne et al. 2017a), including the sub-field of *physics-informed machine learning* that we will explore in more detail in Section 1.3. Some authors even describe the use of data-driven approaches as the beginning of a fourth paradigm in science, following empirical (-1600), theoretical (1600-1950), and computational science (1950-2000) (Hey et al. 2009; Schleder et al. 2019).

As a separate note, many scientists have decided to embark on the enterprise of integrating machine learning in their everyday research due to the apparent success that machine learning has had in recent decades. Not many years ago, domain scientists had often been reluctant to learn about machine learning methods, judging them as opaque black boxes, unreliable, and not respecting domain-established knowledge (Coveney et al. 2016). Even with this caveat, advances in the field of machine learning, and particularly in deep learning, have allowed statistical models to learn at multiple levels of abstraction and capture extremely complex nonlinear patterns and information hidden in large datasets (LeCun et al. 2015). In other words, the fields of mechanistic modelling and statistical modelling have mostly evolved independently (Zdeborová 2020). However, there has been an increasing interest in making mechanistic models more flexible, as well as introducing domain-specific or physical constraints and interpretability in machine learning models.

Hardware. Most of the improvements between 1986 and 2015 that explain the success of deep learning can be attributed, firstly, to the increase in size of available datasets, as we discussed above, and secondly, to the creation of more powerful computers that can handle larger artificial neural networks (Goodfellow et al. 2016). It is impossible to deny the central role that technology plays in the growth of machine learning and, more broadly, the field of artificial intelligence. Emphasizing the argument made in the previous item regarding the different paradigms in science, Thomas Kuhn, a Berkeley professor at the time of publishing his famous book *The Structure of Scientific Revolutions*, identified the introduction of new technologies as one of the most important triggers of a scientific revolution leading to a new scientific paradigm (Kuhn 1962).

However, a hammer by itself does not build a house. The increase alone of computer power does not explain the rapid growth of the scientific computing community, the incorporation of modern programming practices, and introduction of machine learning in the sciences. We still require the software and the tools for scientific computing. As opposed to hardware, there is no silver bullet for software development, meaning that software is an organism whose growth is much slower and more linear than hardware (Brooks Jr 1995). Furthermore, it is important to remark that nowadays machine learning models are being trained on both low and high performance computers. Let us use this point to transition our discussion to the importance of software.

(Open & scientific) Software. Modern scientists have adopted scientific computing as part of their working routine. This includes the use of open-source and community-driven programming languages such as Julia, Python, and R, and the use of enhanced tools like Jupyter that allow interactive computing while still prioritizing the scientific narrative when doing research (Granger et al. 2021). Tools like Jupyter have allowed domain scientists to embrace the *computing* side of *scientific* analysis, no surprise it has been recognized as one of the ten computer programs that revolutionized science (Perkel 2021). These factors have birthed a rich software ecosystem where computing and science combine to produce software that adapts to the needs of scientists, including data preprocessing, exploration and analysis. Let us see some examples of how this works in practice.

Maturity of scientific software. Machine learning applications in the Earth sciences has been empowered in the Python ecosystems by leveraging a rich stack of different packages. (See the *Jupyter Meets the Earth* project website for a full illustration of this: `https://jupytearth.org/jupyter-resources/introduction/ecosystem.html`.) On the bottom of the stack we find the Python core libraries, including Numpy for multidimensional

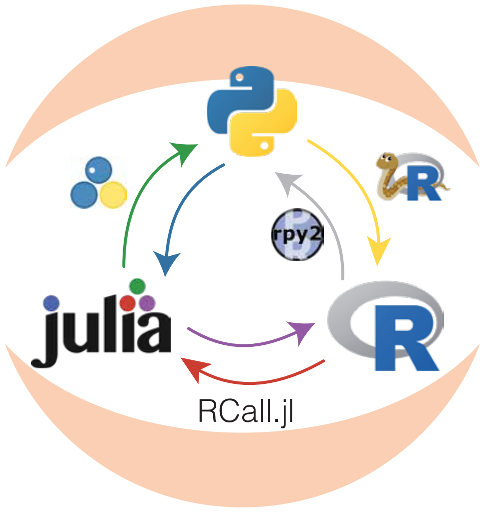


Figure 1.1: *Interoperability between Julia, Python and R. This include the use of Python from Julia with `PyCall.jl` and `PythonCall.jl`; Julia to R with `JuliaCall`; R to Python with `rpy2`; Python to R with `reticulate`; R to Julia with `RCall.jl`; and finally Julia to Python using `PyJulia` and `JuliaCall`.*

arrays (Harris et al. 2020), Matplotlib for plotting (Hunter 2007), Pandas for data frames (McKinney 2010), and Jupyter for interactive computing (Pérez et al. 2007). On top of them, we find more specialized libraries, Scikit-learn being the most popular for machine learning (Pedregosa et al. 2011). We also highlight xarray, a Python package designed to manipulate labeled tensor data such as those found in Earth science datasets (Hoyer et al. 2017a).

Another example that we will further explore in Chapters 2 and 4 is the development of the scientific machine learning ecosystem in Julia. Julia is a recent but mature programming language that has already a large tradition in implementing packages to advance differentiable programming (Bezanson et al. 2017, 2012), with particular emphasis on differential equation solvers (Rackauckas et al. 2016) and sensitivity analysis (Rackauckas et al. 2020). Central to the successful integration of differentiable programming is software interoperability, meaning that the automatic differentiation machinery works when applied on top of numerical solvers and probabilistic programming libraries (Rackauckas et al. 2019).

Outside of the individual realm of each programming language, it is now possible to write software that combines libraries from different programming languages, leveraging the advantages of each one of them. This multilanguage approach is explored in this thesis on the development of `ODINN.jl` (Bolibar et al. 2023b) and `SphereUDE.jl` (Sapienza et al. 2024b), where the core source code is written in Julia but subroutines run processes in Python. Figure 1.1 shows some of the tools allowing communication between Julia, Python, and R.

Cloud computing. In recent years, scientific workflows in Earth and environmental sciences have benefited from transitioning from local to cloud computing (Abernathey et al. 2021; Gentemann et al. 2021; Mesnard et al. 2020). The adoption of cloud computing facilitates the creation of shared computational environments, datasets, and analysis pipelines, which leads to more reproducible scientific results by enabling standalone software containers like Binder (Jupyter et al. 2018) for other researchers to easily reproduce scientific results. On the other hand, working in cloud environments lowers the access threshold for many scientists from under-represented groups across multiple institutions and continents (see `https://coessing.org/2023-school/`). An example of this is the development of CryoCloud, a cloud-computing platform specially dedicated for cryospheric research (Snow et al. 2023).

Adopting software development common practices. The arrival of scientists in computational domains has enriched the domain sciences by incorporating practices generally associated with software developers and designers. Examples of this include better code documentation, which makes software easier to distribute and use, use of version control systems such as git and Github for team development, and testing to ensure trustworthiness and stability of the software (Stoudt et al. 2021). More importantly, the scientific community is recognizing the importance of community work and the ineffectiveness of developing code in isolation (Turk 2013). We hope the next decades see more academics and scholars recognizing the importance of developing community-driven scientific software.

Reproducibility and collaboration. All these previous points build on each other to create an ecosystem where communication and collaboration between pairs is facilitated. The use of good software practices also facilitates the development of scientific results that are computationally reproducible.

Cultural relevance. The Earth is facing a climate crisis that is impossible to deny (Pörtner et al. 2022). There is a call for action to the whole scientific community to come up with ideas and solutions that alleviate the consequences of climate change. One of the key points in this discussion is the role that machine learning will play in the next years, especially for physical model projections, simulations, and data-assimilation methods that can help us understand how the Earth will respond to fast changing temperatures. The scientific community has acknowledged the importance of machine learning and artificial intelligence to tackle climate change (Rolnick et al. 2022), at the same time as the World Economic Forum has identified machine learning as a key element of the modelling of Earth (The World Economic Forum 2018). With the advent of new hybrid approaches that combine machine learning with physical knowledge in the form of differential equations, there is an opportunity to better understand the physical properties of glaciers by leveraging flexible approaches that assimilate observational data. Improving ice flow modeling techniques is a key research area in climate and Earth science, providing an improved understanding of the contribution of glaciers and ice sheets to both water resources and sea-level rise.

1.2 Forward and inverse modelling: the language of scientific discovery

Many of the goals being pursued by the scientific modelling community can be framed as a combination of forward and inverse methods. In statistics, forward modelling is the world of prediction while inverse modelling is the world of inference. Let us explore these two modes of modelling in the context of physical systems and its connection to statistical modelling. The general mathematical framework in this chapter is inspired by the celebrated article of UC Berkeley professor Leo Breiman titled *Statistical Modelling: The Two Cultures* (Breiman 2001).

1.2.1 Forward modelling

Given an input $x \in \mathcal{X}$ and an output $y \in \mathcal{Y}$, our goal is to learn something about the map

$$y = G(x; \beta), \tag{1.1}$$

where β refers to the model parameter(s). The forward map G is usually one of the following:

(i) A known function that is computationally expensive to evaluate.

(ii) An unknown function that describes the behaviour of certain system.

(iii) A partially known function with flexibility provided by the parameter β.

Here the input and output spaces ($\mathcal{X}$ and $\mathcal{Y}$) can be finite vector spaces or infinite dimensional functional spaces, to give a few examples. An important remark here is that the map G potentially includes stochasticity, for example in the form of observational noise, which can be easily modelled as $G = G(x; \beta, \epsilon)$ with ϵ some random variable.

Given data in the form of pairs $\{(x_i, y_i)\}_{i=1}^n$, there are two goals: being able to predict new output responses (prediction) and learn something about the nature of the forward map G (inference or information). Traditional statistical methods have focused their attention to approach (iii), known as the *data modelling culture*, where our goal is to learn the map G by performing inference on the model parameters β. This correspond to cases where we know or assume something about the data generating process of our data.

On the other side, the *algorithmic modelling culture* typically will deal with cases (i) and (ii). Here we assume that the forward map G can be approximated by a new map

$$y^\dagger = G^\dagger_\theta(x; \beta) \tag{1.2}$$

parametrized by an unknown parameter $\theta \in \Theta$. Notice that here we have used the parameter β to refer to parameters in the data generating model, while we will reserve θ for algorithmic parameters. In Section 1.3 we will introduce the field of physical-informed machine learning and see how different families of methods approach the forward modelling.

1.2.1.1 The old recipe for physics: differential equations

Mechanistic or process-based models have played a central role in a wide range of scientific disciplines. They consist of precise mathematical descriptions of physical mechanisms, including the modelling of causal interactions, feedback loops, and dependencies between components of the system under consideration (Rackauckas et al. 2020). These mathematical representations typically take the form of differential equations.

Differential equations are the cornerstone of the physical sciences. For centuries, the scientific community relied only on theoretical and analytical approaches to model and solve systems governed by differential equations. The introduction of numerical methods and, most importantly, the development of the computer, allowed scientists to enter the realm of computational modelling from 1950 to the present day (Hey et al. 2009). With such numerical methods to approximate their solutions, differential equations led to fundamental advances in the understanding and prediction of physical systems.

Similar to statistical and machine learning models, the solution of a differential equation can be seen as a function that maps parameter and initial conditions to state variables. In cases where the forward model is dictated by a differential equation, we have $y = H(u)$, where $u : \Omega \mapsto \mathbb{R}$ is the solution of a differential equation of the form

$$\mathcal{D}_{\text{interior}}(x;\beta)\,u = 0 \qquad \text{dom}(u) = \Omega \tag{1.3}$$

$$\mathcal{D}_{\text{constraint}}(x;\beta)\,u = 0 \qquad \text{dom}(u) = \Gamma \tag{1.4}$$

where $\mathcal{D}_{\text{interior}}$ a differential operator depending on both the input and parameter; $\mathcal{D}_{\text{constraint}}$ is an operator imposing the boundary conditions, initial conditions, and/or constraints on the values the function u takes in the domain Γ; and H is a given function mapping the latent state to observational space (Bryson et al. 1979). Forward models determined by differential equations correspond to cases where the forward map is defined implicitly. A simple example could be solutions of the heat equation, where

$$\mathcal{D}_{\text{interior}}(x;\beta)\,u = \frac{\partial u}{\partial t} - \nabla \cdot (c(\beta)\nabla u) = 0 \quad \text{dom}(u) = [0,\infty) \times V \tag{1.5}$$

$$\mathcal{D}_{\text{constraint}}(x;\beta)\,u = u - u_0(x) = 0 \qquad \text{dom}(u) = (\{0\} \times V) \cup ([0,\infty) \times \partial V) \tag{1.6}$$

where $c(\beta)$ is the diffusivity coefficient, and with initial condition $u = u_0(x)$. Both x and β here can be used for parameters involved in the differential operator or initial/boundary conditions. For the purposes of this thesis, the forward model in glacier modelling is dictated by the mechanistic laws governing ice flow given in the form of partial differential equations arising from approximations of the Navier-Stokes equation (Section 3.1.4)

Solving differential equations is usually carried out by numerical solvers (Hairer et al. 2008; Wanner et al. 1996). We will cover numerical methods for differential equations in the following chapters.

1.2.2 Inverse modelling

Inverse modelling is the world of statistical inference. Provided with a forward model, inverse modelling consists in using observation data $\{(x_i, y_i)\}_{i=1}^n$ to recover the parameters of the forward model that can best explain the data, namely β or θ for the data and algorithmic modelling cultures, respectively. This is performed via the conciliation between model predictions and observations, a process also known as calibration of the model, where optimal parameters and/or initial conditions are found to match the two.

Inverse modelling is one of the ways to bridge the statistical and mechanistic modelling fields (Rüde et al. 2018; Wigner 1960). The field of dynamical data analysis is full of examples where systems of ordinary differential equations are used to model observed data (Ramsay et al. 2017). However, the estimation of model parameters becomes impossible as the number variables and the expressivity of the model increases, especially when considering highly non-linear processes dictated by hidden physics (Karniadakis et al. 2021). Furthermore, for stochastic forward models, the intractability of the likelihood function represents a major challenge for statistical inference (Cranmer et al. 2020). The integration of automatic differentiation and, more broadly, differentiable programming, has provided new tools for resolving complex simulation-based inference problems (Cranmer et al. 2020).

In the context of this thesis, the inverse model or data assimilation pipeline consists of inferring the underlying rheological properties of ice deformation by optimizing a loss function that compares observed glacier velocities with the ones obtained by the forward model (Section 4).

1.2.2.1 The role of differentiable programming

Deep Learning est Mort! Vive Differentiable Programming, Yann LeCun (2018).

Differentiable programming refers to the ability to compute gradients or sensitivities of a model output with respect to model variables or parameters (Shen et al. 2023). Gradients of the forward map given by $\nabla_\beta G(x;\beta)$ can then be used for optimization, sensitivity analysis, Bayesian inference, inverse methods, and uncertainty quantification, within many applications (Razavi et al. 2021). Differentiable programming is a technology that computes gradients involved in probabilistic programming and the efficient evaluation of sensitivities of numerical integrators for differential equations (Blondel et al. 2024). Some authors have recently suggested differentiable programming as the bridge between modern machine learning and traditional scientific models (Gelbrecht et al. 2023; Rackauckas et al. 2021; Ramsundar et al. 2021; Shen et al. 2023). Being able to compute gradients or sensitivities of dynamical systems opens the door to more complex data assimilation models that leverage both strong physical priors and the flexibility to adapt to observations. This is very appealing in fields like computational physics, geophysics, and biology, to mention a few, where there is a broad literature on physical models and a long tradition in numerical methods.

Arguably, the notion of differentiable programming for dynamical systems has a long tradition across the scientific spectrum, including applications in glaciology (Bolibar et al. 2023a; Hascoët et al. 2018; Heimbach et al. 2009; Logan et al. 2020; MacAyeal 1992); fluid dynamics (Giles et al. 2000b; Mohammadi et al. 2009); solid Earth physics (Wu et al. 2023; Zhu et al. 2021b); biology (Strouwen et al. 2022); and design and optimal control (Allaire et al. 2014; Lions 1971; McGreivy et al. 2021; Pironneau 2005). The realization of the importance of differentiable programming in inference and prediction problems involving physical-based modelling lead to the creation of a review paper, which will be the focus of Chapter 2.

Probably the most well-known example of differentiable programming applications in machine learning is the backpropagation algorithm in deep learning (Goodfellow et al. 2016). The backpropagation algorithm is one of the core algorithmic elements that enables the training of complex neural networks. Backpropagation is equivalent to reverse-mode automatic differentiation to compute the gradient of a loss function with respect to the parameters of the neural network, which is later used to perform gradient-based optimization. It is important to remark that differentiable programming is not just a re-branding of the backpropagation algorithms, but it includes a broader set of methods that we will discuss in Chapter 2 with a larger scope of applicability.

1.3 Physics-based machine learning

Roughly defined and a sub-field of machine learning still under development, we can define physics-informed machine learning as the collection of *machine learning techniques that explicitly introduce biases to satisfy certain physical constraints*. These biases can be forced by the design of algorithms that include symmetries, conservation laws, and constraints in the form of differential equations (Karniadakis et al. 2021). For the purpose of this thesis, we are going to assume that these constraints are encoded in the form of differential equations.

Physical constraints come in a full spectrum, whether the physics of the system is known, unknown, or, more interestingly, partially known. This opens the door for data assimilation algorithms that aim to learn governing equations from data. These include methods such as SINDy (Brunton et al. 2016); universal differential equations (UDEs) (Rackauckas et al. 2020); neural ordinary differential equations (Chen et al. 2018; Dandekar et al. 2020); symbolic regression (Chen et al. 2022); Gaussian processes (Chen et al. 2021); physical-informed neural networks (PINNs) (Lagergren et al. 2020; Raissi et al. 2019), including NeuralPDEs (Zubov et al. 2021) and biologically-informed neural networks (Lagergren et al. 2020); and Hamiltonian neural networks (Mattheakis et al. 2020).

Following (Thuerey et al. 2021), we can classify these approaches into the three following classes depending on the goal and the level of compromise between the data and physics-driven cultures:

(i) Surrogate models and emulators

(ii) Soft constraints

(iii) Hard constraints

Surprisingly enough, we are going to see how these three classes align with the three types of assumptions on the forward map G (and $G^\dagger$) that we previously introduced in Section 1.2.1. Let us explore these methods in more detail.

1.3.1 Surrogate models and emulators

The overall goal of surrogate models and emulators is to accelerate the execution of the forward model. This corresponds to the case where the forward map $y = G(x; \beta)$ is known but computationally expensive to run, for example, the numerical solution of a complex differential equation (item (i) in Section 1.2.1). When used as a black box model, surrogates are used as universal approximators of any possible function (item (ii)). Here, we aim to find a surrogate model $y^\dagger = G^\dagger_\theta(x; \beta)$ such that $y \approx y^\dagger$. This is done by fitting the model parameter θ that best approximates the forward map,

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \operatorname{Loss}(y_i, G^\dagger_\theta(x_i; \beta_i)). \tag{1.7}$$

The approximator $G^\dagger$ is usually a very flexible model, typically a deep neural network with customized architectures to adapt to the characteristics of the physical problem. More recent models include neural operators (Kovachki et al. 2021) and Fourier neural operators (Li et al. 2020b) that overcome the discretization-dependence of previous models by learning maps between functional spaces instead of parameters spaces.

These *cheap* versions of the forward map are useful when the evaluation of the forward model is required for many choices of the input variable x but computationally prohibitive with traditional numerical solvers (Cleary et al. 2021). A good example of this is in Bayesian inference (Li et al. 2020b). In more general cases, the emulator can be trained to evaluate the value of likelihood functions, likelihood ratios, or the posterior density (Cranmer et al. 2020; Hermans et al. 2020). Furthermore, it is possible to design hybrid models by combining low fidelity numerical solvers that are cheaper to run with emulators to enhance the resolution of the model (Pestourie et al. 2023).

Nowadays we find physical emulators in practically any scientific discipline dealing with numerical solutions of complex differential equations. These include applications in glaciology (Jouvet et al. 2021), computational fluid dynamics (Kochkov et al. 2021), climate with models like ACE (Watt-Meyer et al. 2023), weather prediction with models like FourCastNet (Pathak et al. 2022) and GraphCast (Lam et al. 2022), quantum chemistry with density functional theory (DFT) emulators (Smith et al. 2017), and high-energy physics (Elvira et al. 2022).

Discovering new representations. Designing models just to achieve predictive results also brings hope for finding new physical representations of complex physical phenomena that are learned by the method itself without the need of inductive biases. A simple example of this consists in using neural network architectures including autoencoders for the forward map in such a way that the latent space encodes meaningful representations of the physical system (Iten et al. 2020).

Inverse modelling with emulators. In the mathematical formulation in Equation (1.7), we have included the parameter β of the forward model. Given observations, this parameter can be inferred using some data assimilation method, for example, via gradient-based optimization involving now gradients of the emulator $\nabla_\beta G_\theta^\dagger(x;\beta)$. A good example of this is the inversion of glacier flow models in (Jouvet 2023) performed on a previously trained emulator (Jouvet et al. 2021).

1.3.2 Soft physical constraints

A recent family of methods that imposed the differential equation structure during training is know as physics-informed neural networks (PINNs) (Raissi et al. 2019). These methods are sometimes called soft constraints because the physical constraint is added in the loss function as a regularization in opposition to the model architecture itself, so the optimization needs to balance the fit to the data and satisfy the differential equation. Given data $\{(x_i, y_i)\}_{i=1}^n$, the solution u of the differential equation embedded in the forward map is approximated by a neural network u_θ with parameter θ such that it minimizes the following combined loss function

$$\sum_{i=1}^{n} \text{Loss}_{\text{data}}(y_i, G_\theta^\dagger(x_i;\beta)) + \lambda \, \text{Loss}_{\text{physics}}(\mathcal{D}(x_i;\beta)\, u_\theta), \tag{1.8}$$

where $\text{Loss}_{\text{data}}$ evaluates the match between observations and the forward model; $\text{Loss}_{\text{physics}}$ penalizes functions u_θ that do not satisfy the differential equation; and $\lambda \geq 0$ is an hyperparameter controlling the contribution of these two to the final loss. The forward map G is then replaced by the algorithmic approximation $G_\theta^\dagger$ as follows:

$$y = G(x;\beta) \qquad y = H(u) \qquad \mathcal{D}_{\text{interior}}(x;\beta)u = 0 \tag{1.9}$$

$$\hat{y} \approx G_\theta^\dagger(x;\beta) \qquad y^\dagger = H(u_\theta) \qquad \mathcal{D}_{\text{interior}}(x;\beta)u_\theta \approx 0. \tag{1.10}$$

The physical loss penalty $\text{Loss}_{\text{physics}}$ may or may not include boundary or initial conditions, providing more flexible constraints than traditional numerical solvers where both need to be specified. In glaciology, PINNs have been used for scientific discovery of rheological (Wang et al. 2023; Wang et al. 2022) and basal sliding properties (Riel et al. 2021); ice sheet modelling (He et al. 2023); and ice-thickness interpolation techniques based on physical constraints for sparse data (Teisberg et al. 2021). However, in general PINNs can be difficult to train and computationally expensive in cases where higher-order derivatives are required (Bettencourt et al. 2019).

Alternative differential equation solvers. As introduced here, PINNs can be used as a method to solve the original differential equation. Together with the family of emulators we explored in the previous section, PINNs belong to the family of approaches that aims to use deep learning to numerically solve differential equations (Han et al. 2018; Hasani et al. 2024; Sirignano et al. 2017; Zhu et al. 2021a). An important consideration is that PINNs tend to be slower than high-fidelity partial differential equation numerical solvers, but offer larger flexibility (Pestourie et al. 2023). Since PINNs can be seen as a numerical solvers, we can still perform inference on the trained solution u_θ with respect to the model parameter β.

Smoothing methods. The loss function in Equation (1.8) equals the empirical loss with a regularization term that imposes solutions that are *smooth* according to a prescribed differential equation. This approaches coincides with profiling estimation methods in the literature of dynamical data analysis (Ramsay et al. 2017). A simpler case of this is smoothing splines, where the penalization term includes second derivatives of the regression function (Green et al. 1993).

1.3.3 Hard physical constraints and universal differential equations

This last family of methods integrates the physics by directly solving the differential equation using a numerical solver. This approach includes universal differential equations (UDEs) (Rackauckas et al. 2020), which in particular include neural differential equation approaches (Chen et al. 2018). More generally, neural differential equations are forward models based on numerical solvers where at least one term inside the differential equation has been replaced by a neural network (Ramadhan et al. 2022). Training is performed via an optimization based on *trajectory matching* (Ramsay et al. 2017):

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \operatorname{Loss}(y_i, G(x_i; \beta_\theta)), \tag{1.11}$$

where the parameter β of the physical model has been replaced by a more general algorithmic representation β_θ with θ the parameter to be optimized. Replacing the model parameter β by a function parametrized by a new (algorithmic) parameter θ may seem confusing, but it is actually very simple and intuitive. Consider the harmonic oscillator

$$\mathcal{D}_{\text{interior}}(x; \beta)\, u = \frac{\partial^2 u}{\partial t^2} + \omega^2 u = 0 \tag{1.12}$$

with natural frequency ω. We can augment this system by allowing $\omega = \omega_\theta(t)$ to be a function of time. This function can be parametrized by a neural network with weights θ that will be trained based on observations.

This approach has the advantage of integrating both data and algorithmic cultures under the same mathematical framework. As noted in their mathematical formulation, we still preserved the original map G (item (iii)) but we add algorithmic flexibility inside the differential equation by allowing the parameter β to be more flexible. In the extreme case of

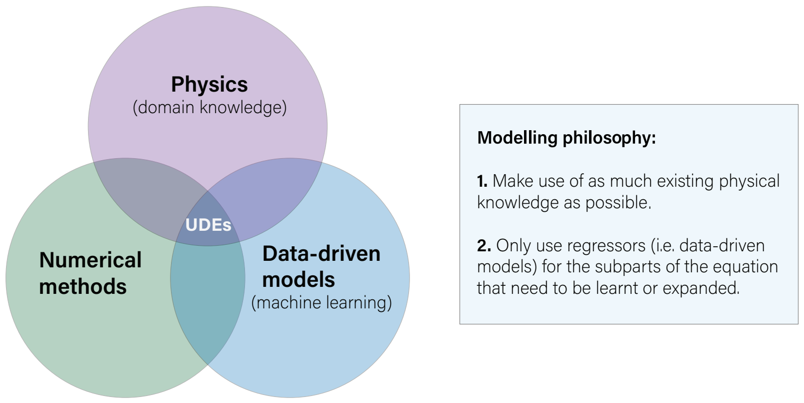

Physics
(domain knowledge)
UDEs
Numerical methods
Data-driven models
(machine learning)
Modelling philosophy:
1. Make use of as much existing physical knowledge as possible.
2. Only use regressors (i.e. data-driven models) for the subparts of the equation that need to be learnt or expanded.

Figure 1.2: *Basic representation of universal differential equations (UDEs) and their associated modelling philosophy. UDEs sit at the intersection of physical domain knowledge, represented by differential equations, numerical methods used to solve the differential equations and data-driven models, often represented as machine learning.*

a completely unknown forward map (item (ii)), we can still use neural differential equations as an universal approximator via the forward map

$$\mathcal{D}_{\text{interior}}(x;\beta)\, u = \frac{\partial u}{\partial t} - f_\theta(x;\beta)(u) = 0, \tag{1.13}$$

where f_θ is a function that described the dynamics of the system parametrized by the parameter θ, including neural networks, polynomial expansions and splines.

Augmented dynamics. The philosophy behind UDEs is to embed a rich family of parametric functions inside a differential equation, so the base structure of the differential equation is preserved but more flexibility is allowed in the model at the moment of fitting observed data. This is particularly useful when our goal is to use as much existing domain knowledge in the form of differential equations as possible and try to learn just the new parts with regressors (see Figure 1.2) (Bolibar et al. 2023a).

Differentiable programming through the solver. The hard physical constraint is imposed by directly solving the differential equation involved in G using numerical solvers. This means that in order to perform gradient-based optimization to solve Equation (1.11) we need to compute the gradient of the numerical solution of a differential equation. As we mentioned in the previous section, this is carried out via differentiable programming tools that we will extensively explore in Chapter 2. Either when the calculation of gradients requires more sophisticated tools, methods based on hard constraints tend to converge after just a few epochs (Bolibar et al. 2023a; Rackauckas et al. 2020; Zhou et al. 2024).

1.3.4 Further remarks

Before finishing this section, let us just cover some short remarks.

From physics to machine learning or from machine learning to physics? Another phrasing of the distinction between soft and hard physical constraint methods is whether we embed machine learning (e.g., neural networks) into existing physical models or we bring physical models into existing machine learning methods. One of the main differences here is what differentiable programming machinery is required in order to train models with many parameters. Incorporating differentiable programming capabilities into existing simulation codes is a more direct way to exploit the advances in deep learning than trying to incorporate domain knowledge into an entirely foreign substrate such as a deep neural network (Cranmer et al. 2020).

Why are we only talking about neural networks? We should not! Almost all algorithms used in the literature use neural networks to parameterize unknown functions. Possibly this is due to the success of neural networks in learning complex non-linear relationships and their easy availability in modern scientific software. However, there is much room for new methods and the use of techniques such as Gaussian processes to combine statistical and physical models. Gaussian processes have been applied in ocean current modelling via the design of kernels that incorporate physical prior information in the form of differential equations (Berlinghieri et al. 2023). Parallelisms between Gaussian process regression and numerical solvers for PDEs have also been recently explored (Chen et al. 2021; Heinonen et al. 2018; Karniadakis et al. 2021; Pförtner et al. 2022).

1.4 Conclusions

There are many challenges in applying machine learning to the Earth sciences (Karpatne et al. 2017b), including the development of geostatistical methods that account for the spatiotemporal nature of the data (Chiles et al. 2012), the validation of statistical models in cases where fundamental assumptions like independence of samples, and independence between training and test set are violated (Hoffimann et al. 2021), and the usage and fitting of forward models dictated by complex dynamics. There is a great opportunity for statisticians interested in working on exciting new problems in Earth sciences. With the rise of remote sensing observations and re-analysis products in the last decades, the Earth sciences has shifted from being a field strongly based on physical models to the realm of big data. Pivoting between the physics and data-driven approaches, there is a unique opportunity to explore the now *reasonable* joint effectiveness of physical models and large amounts of data (Halevy et al. 2009; Wigner 1960).

Chapter 2

Differentiable programming for differential equations

Most of the contents included in this chapter belong to the manuscript in preparation of a review paper on sensitivity methods, tentatively titled

- F. Sapienza, J. Bolibar, F. Schäfer, P. Heimbach, G. Hooker, F. Pérez, P. Persson, C. Rackauckas, V. Boussange, B. Groenke, and A. Pal (2024c). “Differentiable Programming for Differential Equations: A Review”. In: *preparation*

The content of this chapter started as the spin-off of the appendix in (Bolibar et al. 2023a) (Chapter 4) and grew to include a very detailed overview of sensitivity methods for models based on numerical solutions of differential equations. From the first day of this project, all the text and code were openly available in GitHub and contributors/collaborators were invited to participate under the following statement:

> **To the community, by the community.** *This manuscript was conceived with the goal of shortening the gap between developers and practitioners of differentiable programming applied to modern scientific machine learning. With the advent of new tools and new software, it is important to create pedagogical content that allows the broader community to understand and integrate these methods into their workflows. We hope this encourages new people to be an active part of the ecosystem, by using and developing open-source tools. This work was done under the premise* ***open-science from scratch****, meaning all the contents of this work, both code and text, have been in the open from the beginning and that any interested person can contribute to the project. You can contribute directly to the GitHub repository* `github.com/ODINN-SciML/DiffEqSensitivity-Review`.

This work started with three authors and grew to include a total of 12 contributions. This chapter benefited from comments and suggestions of all co-authors.

2.1 Abstract

The differentiable programming paradigm has become a central component of modern machine learning techniques. A long tradition of this paradigm exists in the context of scientific computing, in particular in differential equation-constrained, gradient-based optimization. The recognition of the strong conceptual synergies between inverse methods and machine learning offers the opportunity to lay out a coherent framework applicable to both fields. For models described by differential equations, the calculation of sensitivities and gradients requires careful algebraic and numeric manipulations of the underlying dynamical system. Here, we provide a comprehensive review of existing techniques to compute gradients of numerical solutions of differential equation systems. We first lay out the mathematical foundations of the various approaches and compare them with each other. Second, we delve into the computational considerations and explore the solutions available in modern scientific software.

2.2 Introduction

Evaluating how the value of a function changes with respect to its arguments and parameters plays a central role in optimization, sensitivity analysis, Bayesian inference, inverse methods, and uncertainty quantification, among many (Razavi et al. 2021). Modern machine learning applications require the use of gradients to efficiently exploit the high-dimensional space of parameters to be inferred or learned (e.g., the weights of a neural network). When optimizing an objective function, gradient-based methods (for example, gradient descent and its many variants (Ruder 2016)) are more efficient at finding a minimum and converge faster to them than gradient-free methods. When numerically computing the posterior of a probabilistic model, gradient-based sampling strategies are better at estimating the posterior distribution than gradient-free methods. Hessians further help to improve the convergence rates of these algorithms and can enable uncertainty quantification around parameter values (Bui-Thanh et al. 2012). Furthermore, the *curse of dimensionality* renders gradient-free optimization and sampling methods computationally intractable for most large-scale problems (Oden et al. 2010).

> *A gradient serves as a compass in modern data science: it tells us in which direction in the vast, open ocean of parameters we should move towards in order to increase our chances of success.*

Models based on differential equations arising in simulation-based science, which play a central role in describing the behaviour of systems in natural and social sciences, are not an exception to the rule (Ghattas et al. 2021). The solution of differential equations can be seen as functions that map parameter and initial conditions to state variables, similar to machine learning models (see Section 1.2.1.1). Some authors have recently suggested differentiable programming as the bridge between modern machine learning and traditional scientific models (Gelbrecht et al. 2023; Rackauckas et al. 2021; Ramsundar et al. 2021; Shen et al. 2023). Being able to compute gradients or sensitivities of dynamical systems opens the door to more complex data assimilation models that leverage in strong physical priors at the same time they offer flexibility to adapt to observations. This is very appealing in fields like computational physics, geophysics, and biology, to mention a few, where there is a broad literature on physical models and a long tradition in numerical methods. The first goal of this work is to introduce some of the applications of this emerging technology and to motivate its incorporation for the modelling of complex systems in the natural and social sciences.

Question 1. *What are the scientific applications of differentiable programming for dynamical systems?*

Sensitivity analysis corresponds to any method aiming to calculate how much the output of a function or program changes when we vary one of the function (or model) parameters. This task is performed in different ways by different communities when working with dynamical systems. In statistics, the sensitivity equations enable the computation of gradients of the likelihood of the model with respect to the parameters of the dynamical system, which can be later used for inference (Ramsay et al. 2017). In numerical analysis, sensitivities quantify how the solution of a differential equation fluctuates with respect to certain parameters. This is particularly useful in optimal control theory (Giles et al. 2000b), where the goal is to find the optimal value of some control (e.g. the shape of a wing) that minimizes a given loss function. In recent years, there has been an increasing interest in designing machine learning workflows that include constraints in the form of differential equations. Examples of this include methods that numerically solve differential equations, such as physics-informed neural networks (PINNs, Section 1.3.2) and universal differential equations (UDEs, Section 1.3.3).

However, when working with differential equations, the computation of gradients is not an easy task, both regarding the mathematical framework and software implementation involved. Except for a small set of particular cases, most differential equations require numerical methods to approximate their solution. This means that solutions cannot be directly differentiated and require special treatment to compute first or second-order derivatives. Furthermore, numerical solutions introduce approximation errors. These errors can be propagated and amplified during the computation of the gradient. Alternatively, there is a broad literature on numerical methods for solving differential equations (Hairer et al. 2008; Wanner et al. 1996). Although each method provides different guarantees and advantages depending on the use case, this means that the tools developed to compute gradients when using a solver need to be universal enough in order to be applied to all or at least to a large set of them. As coined by Uwe Naumann, *the automatic generation of optimal (in terms of robustness and efficiency) adjoint versions of large-scale simulation code is one of the great open challenges in the field of High-Performance Scientific Computing* (Naumann 2011). The second goal of this article is to review different methods that exist to achieve this goal.

Question 2. *How to efficiently compute the gradient of a function that depends on the numerical solution of a differential equation?*

The broader set of tools known as automatic or algorithmic differentiation (AD) aims to compute derivatives by sequentially applying the chain rule to the sequence of unit operations that constitute a computer program (Griewank et al. 2008; Naumann 2011). The premise is simple: every computer program is ultimately an algorithm described by a nested concatenation of elementary algebraic operations, such as addition and multiplication, that are individually easy to differentiate and their composition is easy to differentiate by using the chain rule (Giering et al. 1998). More broadly than AD, differentiable programming encapsulates the set of software tools that allows to compute efficient and robust gradients though complex algorithms, including numerical solvers (Innes et al. 2019). Although many modern differentiation tools use AD to some extent, there is also a family of methods that compute the gradient by relying on an auxiliary set of differential equations and/or compute an intermediate adjoint. Furthermore, it is important to be aware than when using AD or any other technique we are differentiating the algorithm used to lead to the numerical solution, no the numerical solution itself, which can lead to wrong results (Eberhard et al. 1996).

The differences between methods to compute sensitivities arise both from their mathematical formulation and their computational implementation. The first provides different guarantees on the method returning the actual gradient or a good approximation thereof. The second involves how theory is translated to software, and what are the data structures and algorithms used to implement it. Different methods have different computational complexities depending on the total number of parameters and size of the differential equation system, and these complexities are also balanced between total execution time and required memory. The third goal of this work, then, is to illustrate the different strengths and weaknesses of these methods, and how to use them in modern scientific software.

Question 3. *What are the advantages and disadvantages of different differentiation methods and how can I incorporate them in my research?*

Differentiable programming is opening new ways of doing research across sciences. Arguably, its potential has so far been under-explored but is being rediscovered in the age of data-driven science. In order to realize its full potential, we need close collaboration between domain scientists, methodological scientists, computational scientists, and computer scientists in order to develop successful, scalable, practical, and efficient frameworks for real world applications. As we make progress in the use of these tools, new methodological questions start to emerge. How do these methods compare? How can they be improved? In this review we present a comprehensive list of the methods that exists in the intersection of differentiable programming and differential equation modelling.

A full discussion of the first of our questions emphasizing the importance of gradients of solutions of ODEs in a variety of scientific domains, covering computational fluid dynamics, geosciences, meteorology, oceanograpgy, climate science, glaciology, ecology, and biology, is

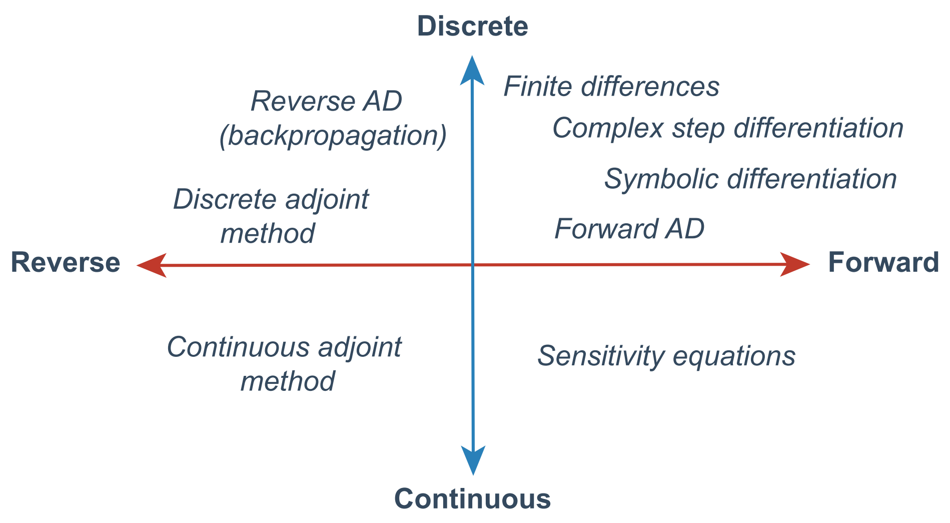


Figure 2.1: *Schematic representation of the different methods available for differentiation involving differential equation solutions. These can be classified depending if they find the gradient by solving a new system of differential equations (continuous) or if instead they manipulate unit algebraic operations (discrete). Additionally, these methods can be categorized based on their alignment with the direction of the numerical solver. If they operate in the same direction as the solver, they are referred to as forward methods. Conversely, if they function in the opposite direction, they are known as reverse methods.*

included in the full manuscript of the review paper available in the GitHub repository and it was omitted in this thesis. The review paper is structured in three main sections, looking at differentiable programming for differential equations from three different perspectives: a domain science perspective, a mathematical perspective (Section 2.3) and a computer science perspective (Section 2.4).

2.3 Methods: A mathematical perspective

There is a large family of methods for computing gradients and sensitivities of systems of differential equations. Depending on the number of parameters and the complexity of the differential equation we are trying to solve, they have different mathematical, numerical, and computational advantages. These methods can be roughly classified as follows (Ma et al. 2021a):

- *Continuous* vs *discrete* methods
- *Forward* vs *reverse* methods

Figure 2.1 displays a classification of some methods under this two-fold division.

The *continuous* vs *discrete* distinction is one of mathematical and numerical nature. When solving for the gradient of a differential equation, one needs to derive both a mathematical expression for the gradient (the differentiation step) and solve the equations using a numerical solver (the discretization step) (Bradley 2013; Onken et al. 2020; Sirkes et al. 1997; Zhang et al. 2014). Depending on the order of these two operations, we are going to talk about discrete methods (discretize-then-differentiate) or continuous methods (differentiate-then-discretize). In the case of *discrete* methods, gradients are computed based on simple function evaluations of the solutions of the numerical solver (finite differences, complex step differentiation) or by manipulation of atomic operations inside a numerical solver (AD, symbolic differentiation, discrete adjoint method). In the case of *continuous* methods, a new set of differential equations is derived for the sensitivity (sensitivity equations) or the adjoint (continuous adjoint method) of the system, both quantities that allow the calculation of the desired gradient. When comparing between discrete and continuous methods, more than talking about computational efficiency we are focusing on the mathematical consistency of the method, that is, *is the method estimating the right gradient?*. Discrete methods compute the exact derivative of the numerical approximation to the loss function, but they do not necessarily yield to an approximation of the exact derivatives of the objective function ((Walther 2007), Section 2.3.9.3).

The *forward* vs *reverse* distinction regards when the gradient is computed, if this happens during the forward pass of the numerical solver or in a later recalculation (Griewank et al. 2008). In all *forward* methods the solution of the differential equation is solved sequentially and simultaneously with the gradient during the forward pass of the numerical solver. On the contrary, *reverse* methods compute the gradient tracking backwards the forward model by resolving a new problem that moves in the opposite direction as the original numerical solver. For systems of ordinary differential equations (ODEs) and initial value problems (IVPs), most numerical methods solve the differential equation progressively moving forward in time, meaning that reverse methods then solve for the gradient moving backwards in time.

As we will discuss in the following sections, forward methods are very efficient for problems with a small number of parameters we want to differentiate with respect to, while backwards methods are more efficient for a large number of parameters but they come with a larger memory cost which needs to be overcome using different performance tricks. With the exception of finite differences and complex step differentiation, the rest of the forward methods (i.e. forward AD, sensitivity equations, symbolic differentiation) compute the full sensitivity of the differential equation, that is, how the full solution of the ODEs changes when we change the parameters of the model. This can be computationally expensive for large systems. Conversely, reverse methods are based on the computation of intermediate variables, known as the adjoint or dual variables, that cleverly avoid the unnecessary calculation of the full sensitivity at expenses of larger memory cost (Givoli 2021). For this reason, reverse methods can be also labeled as adjoint methods (Ma et al. 2021a).

The rest of this section is organized as follows. We will first introduce some basic mathematical notions that are going to facilitate the discussion of the sensitivity methods (Section 2.3.1). Then we will embark in the mission of mathematically introducing each one of methods listed in Figure 2.1. We will finalize the discussion in Section 2.3.9 with an comparison of some of mathematical foundations of these methods.

2.3.1 Preliminaries

Consider a system of first order ordinary differential equations (ODEs) given by

$$\frac{du}{dt} = f(u, \theta, t), \tag{2.1}$$

where $u \in \mathbb{R}^n$ is the unknown solution; $f : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R} \mapsto \mathbb{R}^n$ is a function that depends on the state u, some vector parameter $\theta \in \mathbb{R}^p$, and potentially the independent variable t which we will refer as time; and with initial condition $u(t_0) = u_0$. Here n denotes the total number of ODEs and p the dimension of a parameter embedded in the functional form of the differential equation. Although we here consider the case of ODEs, that is, when the derivatives are just with respect to the time variable t, the ideas presented here can be extended to the case of partial differential equations (PDEs; for example, via the method of lines (Ascher 2008)) and differential algebraic equations (Wanner et al. 1996). Except for a minority of functions $f(u, \theta, t)$, solutions to Equation (2.1) need to be computed using a numerical solver.

2.3.1.1 Numerical solvers for ordinary differential equations

Numerical solvers for the solution of ODEs or initial value problems (IVP) can be classified as one-step methods, among which Runge-Kutta methods are the most widely used, and multi-step methods (Hairer et al. 2008). Given an integer s, a s-stage Runge-Kutta method is defined by generalizing numerical integration quadrature rules as follows

$$\begin{aligned} u^{n+1} &= u^n + \Delta t_n \sum_{i=1}^{s} b_i k_i \\ k_i &= f\left(u^n + \sum_{j=1}^{s} a_{ij} k_j,\, \theta,\, t_n + c_i \Delta t_n \right) \qquad i = 1, 2, \ldots, s. \end{aligned} \tag{2.2}$$

where $u^n \approx u(t_n)$ approximates the solution at time t_n; timesteps $\Delta t_n = t_{n+1} - t_n$; and coefficients a_{ij}, b_i, and c_j, with $i, j = 1, 2, \ldots, j$, usually represented in the form of a tableau. A Runge-Kutta method is called explicit if $a_{ij} = 0$ for $i \leq j$; diagonally implicit if $a_{ij} = 0$ for $i < j$; and implicit otherwise. Different choices of number of stages and coefficients give different orders of convergence of the numerical scheme (Butcher 2001; Butcher et al. 1996).

On the contrary, multisteps linear solvers are of the form

$$\sum_{i=0}^{k_1} \alpha_{ni} u^{n-i} = \Delta t_n \sum_{j=0}^{k_2} \beta_{nj} f(u^{n-j}, \theta, t_{n-j}) \tag{2.3}$$

where α_{ni} and β_{nj} are numerical coefficients (Hairer et al. 2008). In most cases, including Adam and BFG methods, we have the coefficients $\alpha_{ni} = \alpha_i$ and $\beta_{nj} = \beta_j$, meaning that the coefficients do not depend on the iteration. Notice that multisteps linear methods are linear in the function f, which is not the case in Runge-Kutta methods with intermediate evaluations (Ascher 2008). Explicit methods are characterized by $\beta_{n,0} = 0$ and are easy to solve by direct iterative updates. For implicit methods, the usually non-linear equation

$$g_i(u_i;\theta) = u_i - h\beta_{n0} f(u_i, \theta, t_i) - \alpha_i = 0, \tag{2.4}$$

with α_i a computed coefficient that includes the information of all the past iterations, can be solved using predictor-corrector methods (Hairer et al. 2008) or iteratively using Newton's method (Hindmarsh et al. 2005).

There are many considerations at the moment of picking a numerical solver. One of the most important ones is the stiffness of the differential equation we are trying to solve. Although stiffness is a known phenomena in the study of differential equation solver, different definitions and types of instability exist in the literature. This is due to historical reasons (Dahlquist 1985) as well as the fact that different stiff equations suffer from different types of instabilities. Among them we select the following:

- Stiff equations are equations for which explicit methods do not work and implicit methods work better (Wanner et al. 1996).
- Stiff differential equations are characterized by dynamics with different time scales (Kim et al. 2021), also characterized by the phenomena of increasing oscillations (Dahlquist 1985).

Stability properties can be achieved by the use of implicit methods over explicit methods. When using explicit methods, smaller timesteps may be required to guaranteed stability.

Another important consideration is how to pick the time-steps Δt_i in a numerical solver (Hairer et al. 2008). Modern solvers include stepsize controllers that pick Δt_i as large as possible to minimize the total number of steps at the same time that they control for large errors in the numerical solution controlled by adjustable relative and absolute tolerances.

2.3.1.2 What to differentiate and why?

We are interested in computing the gradient of a given function $L(u(\cdot, \theta))$ with respect to the parameter θ. This formulation is very general and allows to include many different applications, including the following.

- **Loss function and empirical risk function**. This is usually a real-valued function that quantifies the level of agreement between the model prediction and observations. Examples of loss functions include the squared error

$$L(\theta) = \frac{1}{2} \| u(t_1;\theta) - u^{\text{target}}(t_1) \|_2^2, \tag{2.5}$$

where $u^{\text{target}}(t_1)$ is the desired target observation at some later time t_1. More generally, we can evaluate the loss function at points of the time series for which we have observations,

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \omega_i \, \| u(t_i;\theta) - u^{\text{target}}(t_i) \|_2^2, \tag{2.6}$$

with ω_i some arbitrary non-negative weights. More generally, misfit functions used in optimal estimation and control problems map from the model's state space, in this case the solution $u(t)$, to the observation space define by a new variable $y(t) = H(u(t,\theta))$, where $H : \mathbb{R}^n \mapsto \mathbb{R}^o$ is a given function mapping the latent state to observational space (Bryson et al. 1979). In these cases, the loss function is instead

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \omega_i \, \| H(u(t_i;\theta)) - y^{\text{target}}(t_i) \|_2^2. \tag{2.7}$$

We can also consider the continuous evaluated loss function of the form

$$L(u(\cdot,\theta)) = \int_{t_0}^{t_1} h(u(t;\theta),\theta) dt, \tag{2.8}$$

with h being a function that quantifies the contribution of the error term at every time $t \in [t_0, t_1]$. Defining a loss function where just the empirical error is penalized is known as trajectory matching (Ramsay et al. 2017). Other methods like gradient matching and generalized smoothing the loss depends on smooth approximations of the trajectory and their derivatives.

- **Likelihood function.** From a statistical perspective, it is common to assume that observations correspond to noisy observations of the underlying dynamical system, $y_i = H(u(t_i;\theta)) + \varepsilon_i$, with ε_i errors or residual that are independent of each other and of the trajectory $u(\cdot;\theta)$ (Ramsay et al. 2017). When H is the identity, each y_i corresponds to the noise observation of the state $u(t_i;\theta)$. If $p(Y|t,\theta)$ is the probability distribution of $Y = (y_1, y_2, \ldots, y_N)$, maximum likelihood estimation consists in finding the maximum a posteriori (MAP) estimate of the parameter θ as

$$\theta^* = \underset{\theta}{\text{argmax}} \; \ell(Y|\theta) = \prod_{i=1}^{n} p(y_i|\theta, t_i). \tag{2.9}$$

When $\varepsilon_i \sim N(0, \sigma_i^2 \, \mathbb{I})$ is the isotropic multivariate normal distribution, the maximum likelihood principle is the same as minimizing $-\log \ell(Y|\theta)$ which coincides with the mean squared error of Equation (2.7) (Hastie et al. 2009),

$$\theta^* = \underset{\theta}{\text{argmin}} \; \{ -\log \ell(Y|\theta) \} = \underset{\theta}{\text{argmin}} \sum_{i=1}^{N} \frac{1}{2\sigma_i^2} \, \| y_i - H(u(t_i;\theta)) \|_2^2. \tag{2.10}$$

Provided with a prior distribution $p(\theta)$ for the parameter θ, we can further compute a posterior distribution for θ given the observations Y following Bayes theorem (Murphy 2022). In practice, the posterior is difficult to evaluate and needs to be approximated using Markov chain Monte Carlo (MCMC) sampling methods (Gelman et al. 2013). Being able to further compute gradients of the likelihood allows to design more efficient sampling methods, such as Hamiltonian MCMC (Betancourt 2017).

- **Quantity of interest.** Another important example is when L returns the value of the solution at one or many points, which is useful when we want to know how the solution itself changes as we move the parameter values.
- **Diagnosis of the solution.** In many cases we are interested in optimizing the value of some variable that is a function of the solution of a differential equation. This is the case in design control theory, a popular approach in aerodynamics modelling where goals include maximizing the speed of an airplane or the lift of a wing given the solution of the flow equation for a given geometry profile (Giles et al. 2000a; Jameson 1988; Mohammadi et al. 2004).

In the rest of the manuscript we will use letter L to emphasize that in many cases this will be a loss function, but without loss of generality this includes the richer class of functions included in the previous examples.

2.3.1.3 Sensitivity matrix

In the general case, we are going to work with loss functions of the form $L(\theta) = L(u(\cdot, \theta), \theta)$. Using the chain rule we can derive

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial \theta} + \frac{\partial L}{\partial \theta}. \tag{2.11}$$

The two partial derivatives of the loss function on the right-hand side are usually easy to evaluate. For example, for the loss function in Equation (2.5) this are simply given by

$$\frac{\partial L}{\partial u} = u - u^{\text{target}}(t_1) \qquad \frac{\partial L}{\partial \theta} = 0. \tag{2.12}$$

Just as in this last example, in most applications the loss function $L(\theta)$ will depend on θ just through u, meaning $\frac{\partial L}{\partial \theta} = 0$. The complicated term to compute is the matrix of derivatives $\frac{\partial u}{\partial \theta}$, usually referred to as the *sensitivity* s, and represents how much the full solution u varies as a function of the parameter θ,

$$s = \frac{\partial u}{\partial \theta} = \begin{bmatrix} \frac{\partial u_1}{\partial \theta_1} & \cdots & \frac{\partial u_1}{\partial \theta_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_n}{\partial \theta_1} & \cdots & \frac{\partial u_n}{\partial \theta_p} \end{bmatrix} \in \mathbb{R}^{n \times p}. \tag{2.13}$$

The sensitivity s defined in Equation (2.13) is what is called a *Jacobian*, that is, a matrix of first derivatives for general vector-valued functions. Some of the methods we will discuss here will directly compute the sensitivity, while others will only deal with Jacobian-vector products (JVPs) of the form $\frac{\partial u}{\partial \theta} v$, for some vector $v \in \mathbb{R}^p$. The product $\frac{\partial u}{\partial \theta} v$ is the directional derivative of the function $u(\theta)$, also known as the Gateaux derivative of $u(\theta)$ in the direction v, given by

$$\frac{\partial u}{\partial \theta} v = \lim_{h \to 0} \frac{u(\theta + hv) - u(\theta)}{h}, \tag{2.14}$$

representing how much the function u changes when we perturb θ in the direction of v.

2.3.2 Finite differences

The simplest way of evaluating a derivative is by computing the difference between the evaluation of the function at a given point and a small perturbation of the function. In the case of the function $L : \mathbb{R}^p \mapsto \mathbb{R}$, we can approximate

$$\frac{dL}{d\theta_i}(\theta) = \frac{L(\theta + \varepsilon e_i) - L(\theta)}{\varepsilon} + \mathcal{O}(\varepsilon), \tag{2.15}$$

with e_i the i-th canonical vector and ε the stepsize. Even better, the centered difference scheme leads to

$$\frac{dL}{d\theta_i}(\theta) = \frac{L(\theta + \varepsilon e_i) - L(\theta - \varepsilon e_i)}{2\varepsilon} + \mathcal{O}(\varepsilon^2). \tag{2.16}$$

While Equation (2.15) gives the derivative to an error of magnitude $\mathcal{O}(\varepsilon)$, the centered differences schemes improves the accuracy to $\mathcal{O}(\varepsilon^2)$ (Ascher et al. 2011). Further finite difference stencils of higher order exist in the literature (Fornberg 1988).

However, there are a series of problems associated with finite differences. The first one is due to how it scales with the dimension p of parameter vector θ. Each directional derivative requires the evaluation of the loss function L twice. For the centered differences approach in Equation (2.16), this requires a total of $2p$ function evaluations which demands solving the differential equation each time for a new set of parameters. A second problem is due to rounding errors. Every computer ultimately stores and manipulates numbers using floating point arithmetic (Goldberg 1991). Equations (2.15) and (2.16) involve the subtraction of two numbers that are very close to each other, which leads to large cancellation errors for small values of ε that are amplified by the division by ε. On the other hand, large values of the stepsize give inaccurate estimations of the gradient. Finding the optimal value of ε that balances these two effects is sometimes known as the *stepsize dilemma*, for which algorithms based on prior knowledge of the function to be differentiated or algorithms based on heuristic rules have been introduced (Barton 1992; Hindmarsh et al. 2005; Mathur 2012). Although analytical functions, like polynomials and trigonometric functions, can be computed with machine precision, numerical solutions of differential equations have errors that are typically larger than machine precision, which leads to inaccurate estimations of the gradient when ε is too small. We will further emphasize this point in Section 2.4.1.1.

Despite all these caveats, finite differences can be useful when computing Jacobian-vector products (JVPs). Given a Jacobian matrix $J = \frac{\partial f}{\partial u}$ (or the sensitivity $s = \frac{\partial u}{\partial \theta}$) and a vector v, the product Jv corresponding to the directional derivative and can be approximated as

$$Jv \approx \frac{f(u + \varepsilon v, \theta, t) - f(u, \theta, t)}{\varepsilon}. \tag{2.17}$$

This approach is used in numerical solvers based on Krylov methods, where linear systems are solved by iteratively solving matrix-vectors products (Ipsen et al. 1998).

2.3.3 Automatic differentiation

Automatic differentiation (AD) is a technology that generates new code representing derivatives of a given parent code. Examples are code representing the tangent linear or adjoint operator of the parent code (Griewank et al. 2008). The names *algorithmic* and *computational* differentiation had also been used in the literature, emphasizing the algorithmic rather than automatic nature of AD (Griewank et al. 2008; Margossian 2018). The basis of all AD systems is the notion that complicated functions included in any computer program can be reduced to a sequence of simple algebraic operations that have straightforward derivative expressions, based upon elementary rules of differentiation (Juedes 1991). The derivatives of the outputs of the computer program (dependent variables) with respect to their inputs (independent variables) are then combined using the chain rule. One advantage of AD systems is to automatically differentiate programs that include control flow, such as branching, loops or recursions. This is because any program can be reduced to a trace of input, intermediate and output variables (Baydin et al. 2017).

Depending if the concatenation of these gradients is done as we execute the program (from input to output) or in a later instance where we trace-back the calculation from the end (from output to input), we refer to *forward* or *reverse* AD, respectively. Neither forward nor reverse mode is more efficient in all cases (Griewank 1989), as we will discuss in Section 2.3.3.3.

2.3.3.1 Forward mode

Forward mode AD can be implemented in different ways depending on the data structures we use at the moment of representing a computer program. Examples of these data structures include dual numbers and computational graphs (Baydin et al. 2017).

2.3.3.1.1 Dual numbers Dual numbers extend the definition of a numerical variable that takes a certain value to also carry information about its derivative with respect to a certain parameter (Clifford 1871). We define a dual number based on two variables: a *value* coordinate x_1 that carries the value of the variable and a *derivative* (also known as partial or tangent) coordinate x_2 with the value of the derivative $\frac{\partial x_1}{\partial \theta}$. Just as complex number, we can represent dual numbers as an ordered pair (x_1, x_2), sometimes known as Argand pair, or in the rectangular form

$$x_\epsilon = x_1 + \epsilon \, x_2, \tag{2.18}$$

where ϵ is an abstract number called a perturbation or tangent, with the properties $\epsilon^2 = 0$ and $\epsilon \neq 0$. This last representation is quite convenient since it naturally allow us to extend algebraic operations, like addition and multiplication, to dual numbers (Karczmarczuk 1998). For example, given two dual numbers $x_\epsilon = x_1 + \epsilon x_2$ and $y_\epsilon = y_1 + \epsilon y_2$, it is easy to derive using the fact $\epsilon^2 = 0$ that

$$x_\epsilon + y_\epsilon = (x_1 + y_1) + \epsilon \, (x_2 + y_2) \qquad x_\epsilon y_\epsilon = x_1 y_1 + \epsilon \, (x_1 y_2 + x_2 y_1). \tag{2.19}$$

From these last examples, we can see that the derivative component of the dual number carries the information of the derivatives when combining operations. For example, suppose than in the last example the dual variables x_2 and y_2 carry the value of the derivative of x_1 and x_2 with respect to a parameter θ, respectively.

Intuitively, we can think of ϵ as being a differential in the Taylor series expansion, fact that we can observe in how the output of any scalar functions is extended to a dual number output:

$$\begin{aligned} f(x_1 + \epsilon x_2) &= f(x_1) + \epsilon \, x_2 \, f'(x_1) + \epsilon^2 \cdot (\ldots) \\ &= f(x_1) + \epsilon \, x_2 \, f'(x_1). \end{aligned} \tag{2.20}$$

When computing first order derivatives, we can ignore everything of order ϵ^2 or larger, which is represented in the condition $\epsilon^2 = 0$. This implies that we can use dual numbers to implement forward AD through a numerical algorithm. In Section 2.4.1.2.1 we will explore how this is implemented.

Multidimensional dual number generalize dual number to include a different dual variable ϵ_i for each variable we want to differentiate with respect to (Neuenhofen 2018; Revels et al. 2016). A multidimensional dual number is then defined as $x_\epsilon = x + \sum_{i=1}^p x_i \epsilon_i$, with the property that $\epsilon_i \epsilon_j = 0$ for all pairs i and j. Incorrect implementations of this aspect can lead to *perturbation confusion*, an existing problem in some AD software where dual variables corresponding to different variables result indistinguishable, especially in the case of nested functions (Manzyuk et al. 2019; Siskind et al. 2005). This problem can be further been overcome by computing the full gradient as the combination of independent directional derivatives (see Section 2.3.3.3) Another extension of dual numbers that should not be confused with multidimensional dual numbers are hyper-dual numbers, which allow to compute higher-order derivatives of a function (Fike 2013).

2.3.3.1.2 Computational graph A useful way of representing a computer program is via a computational graph with intermediate variables that relate the input and output variables. Most scalar functions of interest can be represented as a acyclic directed graph with nodes associated to variables and edges to atomic operations (Griewank 1989; Griewank et al. 2008), known as Kantorovich graph (Kantorovich 1957) or its linearized representation via a Wengert trace/tape (Bauer 1974; Griewank et al. 2008; Wengert 1964). We can define $v_{-p+1}, v_{-p+2}, \dots, v_0 = \theta_1, \theta_2, \dots, \theta_p$ the input set of variables; $v_1, \dots, v_{m-1}$ the set of all the intermediate variables, and finally $v_m = L(\theta)$ the final output of a computer program. This can be done in such a way that the order is strict, meaning that each variable v_i is computed just as a function of the previous variables v_j with $j < i$. Once the graph is constructed, we can compute the derivative of every node with respect to other (a quantity known as the tangent) using the Bauer formula (Bauer 1974; Oktay et al. 2020):

$$\frac{\partial v_j}{\partial v_i} = \sum_{\substack{\text{paths } w_0 \to w_1 \to \dots \to w_K \\ \text{with } w_0 = v_i, w_K = v_j}} \prod_{k=0}^{K-1} \frac{\partial w_{k+1}}{\partial w_k}, \tag{2.21}$$

where the sum is calculated with respect to all the directed paths in the graph connecting the input and target node. Instead of evaluating the last expression for all possible paths, a simplification is to increasingly evaluate $j = 1, \dots, m$ using the recursion

$$\frac{\partial v_j}{\partial v_i} = \sum_{w \text{ such that } w \to v_j} \frac{\partial v_j}{\partial w} \frac{\partial w}{\partial v_i} \tag{2.22}$$

Since every variable node w such that $w \to v_j$ is an edge of the computational graph have index less than j, we can iterate this procedure as we run the computer program and solve for both the function and its gradient. This is possible because in forward mode the term $\frac{\partial w}{\partial v_i}$ has been computed in a previous iteration, while $\frac{\partial v_j}{\partial w}$ can be evaluated at the same time the node v_j is computed based on only the value of the parent variable nodes. The only requirement for differentiation is being able to compute the derivative/tangent of each edge/primitive and combine these using the recursion defined in Equation (2.22).

2.3.3.2 Reverse mode

Reverse mode AD is also known as the adjoint of cotangent linear mode, or backpropagation in the field of machine learning. The reverse mode of automatic differentiation has been introduced in different contexts (Griewank 2012) and materializes the observation made by Phil Wolfe that if the chain rule is implemented in reverse mode, then the ratio between the computational cost of the gradient of a function and the function itself can be bounded by a constant that does not depend of the number of parameters to differentiate (Griewank 1989; Wolfe 1982), a point known as the *cheap gradient principle* (Griewank 2012). Given a directed graph of operations defined by a Wengert list, we can compute gradients of any given function in the same fashion as Equation (2.22) but in reverse mode as

$$\bar{v}_i = \frac{\partial \ell}{\partial v_i} = \sum_{w \text{ such that } v_i \to w} \frac{\partial w}{\partial v_i} \bar{w}. \tag{2.23}$$

In this context, the notation $\bar{w} = \frac{\partial \ell}{\partial w}$ is introduced to signify the partial derivative of the output variable, here associated to the loss function, with respect to input and intermediate variables. This derivative is often referred to as the adjoint, dual, or cotangent, and its connection with the discrete adjoint method will be made more explicitly in Section 2.3.9.2.

Since in reverse-mode AD the values of $\bar{w}$ are being updated in reverse order, in general we need to know the state value of all the argument variables v of w in order to evaluate the terms $\frac{\partial w}{\partial v}$. These state values (required variables) need to be either stored in memory during the evaluation of the function or recomputed on the fly in order to be able to evaluate the derivative. Checkpointing schemes exist to limit and balance the amount of storing versus recomputation (see section 2.4.1.2.3).

2.3.3.3 AD connection with JVPs and VJPs

When working with unit operations that involve matrix operations dealing with vectors of different dimensions, the order in which we apply the chain rule matters (Giering et al. 1998). When computing a gradient using AD, we can encounter vector-Jacobian products (VJPs) or Jacobian-vector products (JVP). As their name indicates, the difference between them is that the quantity we are interested in is described by the product of a Jacobian times a vector on the left side (VJP) or the right (JVP). Furthermore, both forward and reverse AD can be thought as a way of computing directional derivatives associated with JVPs (see Equation (2.14)) and VJPs, respectively. In other words, given a function $g : \mathbb{R}^{d_1} \mapsto \mathbb{R}^{d_2}$ that is evaluated during the forward mode of given program, AD will carry terms of the form $Dh(x) \cdot \dot{x}$ (JVP) in forward mode and $\bar{y}^T \cdot Dh(x)$ (VJP) in reverse mode (Griewank et al. 2008).

Let us consider for example the case of a nested loss function $L : \mathbb{R}^p \mapsto \mathbb{R}$ taking a total of p arguments as inputs that can be decomposed as $L(\theta) = \ell \circ g_k \circ \ldots \circ g_2 \circ g_1(\theta)$, with $\ell : \mathbb{R}^{d_k} \mapsto \mathbb{R}$ the final evaluation of the loss function after we apply in order a sequence of intermediate functions $g_i : \mathbb{R}^{d_{i-1}} \mapsto \mathbb{R}^{d_i}$, where we define $d_0 = p$ for simplicity. The final gradient is computed as the chain product of vectors and Jacobians as

$$\nabla_\theta L = \nabla \ell \cdot Dg_k \cdot Dg_{k-1} \cdot \ldots \cdot Dg_2 \cdot Dg_1, \tag{2.24}$$

with Dg_i the Jacobian of each nested function evaluated at the intermediate values $g_{i-1} \circ g_{i-2} \circ \ldots \circ g_i(\theta)$. Notice that in the last equation $\nabla \ell \in \mathbb{R}^{d_k}$ is a vector. In order to compute $\nabla_\theta L$, we can solve the multiplication starting from the right side, which will correspond to multiplying the Jacobians forward from Dg_1 to Dg_k, or from the left side, moving backwards. The important aspect of the backwards case is that we will always be computing VJPs, since $\nabla \ell$ is a vector. Since VJPs are easier to evaluate than full Jacobians, the reverse mode will in general be faster when $1 \ll p$. This example is illustrated in Figure 2.2. For general rectangular matrices $A \in \mathbb{R}^{d_1 \times d_2}$ and $B \in \mathbb{R}^{d_2 \times d_3}$, the cost of the matrix multiplication AB is $\mathcal{O}(d_1 d_2 d_3)$. It is worth noticing that while more efficient methods for matrix-matrix multiplication based on Strassen's recursive algorithm and its variants exist, these are not extensively used in most scientific applications (Huang et al. 2016; Silva et al. 2018). This implies that forward AD requires a total of

$$d_2 d_1 p + d_3 d_2 p + \ldots + d_k d_{k-1} p + d_k p = \mathcal{O}(kp) \tag{2.25}$$

operations, while backwards mode AD requires

$$d_k d_{k-1} + d_{k-1} d_{k-2} + \ldots + d_2 d_1 + d_1 p = \mathcal{O}(k+p) \tag{2.26}$$

operations.

In the general case of a function $L : \mathbb{R}^p \mapsto \mathbb{R}^q$ with multiple outputs and a total of k intermediate functions, the cost of forward AD is $\mathcal{O}(pk+q)$ and the cost of reverse is $\mathcal{O}(p+kq)$. When the function to differentiate has a larger input space than output ($q \ll p$), AD in reverse mode is more efficient as it propagates the chain rule by computing VJPs, the reason why reverse-mode AD is more used in modern machine learning. However, notice that backwards mode AD requires us to save intermediate variables through the forward run in order to run backwards afterwards (Bennett 1973), leading to performance overhead that makes forward AD more efficient when $p \lesssim q$ (Baydin et al. 2017; Griewank 1989; Margossian 2018). In other words, reverse AD is really more efficient when $q \ll p$.

In a practical sense, many AD systems are reduced to the computation of only directional derivatives (VJPs) and JVPs (Griewank et al. 2008). Full Jacobians $J \in \mathbb{R}^{n \times p}$ (e.g., the sensitivity $s = \frac{\partial u}{\partial \theta} \in \mathbb{R}^{n \times p}$) can be fully reconstructed by the independent computation of the p columns of J via the JVPs Je_i, with $e_i \in \mathbb{R}^p$ the canonical vectors; or by the calculation of the m rows of J via the VJPs $e_j^T J$, with $e_j \in \mathbb{R}^n$. An important observation here is then how to efficiently compute sparse Jacobians, which are commonplace in large-scale nonlinear systems and discretized PDEs. For cases with known sparsity pattern, *colored AD* can be used to chunk multiple JVPs or VJPs using the colored Jacobian (Gebremedhin et al. 2005).

2.3.4 Complex step differentiation

An alternative to finite differences that avoids subtractive cancellation errors is based on complex variable analysis. The first proposals originated in 1967 using the Cauchy integral theorem involving the numerical evaluation of a complex-valued integral (Lyness 1967; Lyness et al. 1967). A new approach recently emerged that uses the complex generalization of a real function to evaluate its derivatives (Martins et al. 2003; Squire et al. 1998). Assuming that the function $L(\theta)$ admits a holomorphic extension (that is, it can be extended to a complex-valued function that is analytical and differentiable (Stein et al. 2010)), the Cauchy-Riemann conditions can be used to evaluate the derivative with respect to one single scalar parameter $\theta \in \mathbb{R}$ as

$$\frac{dL}{d\theta} = \lim_{\varepsilon \to 0} \frac{\operatorname{Im}(L(\theta + i\varepsilon))}{\varepsilon}, \tag{2.27}$$

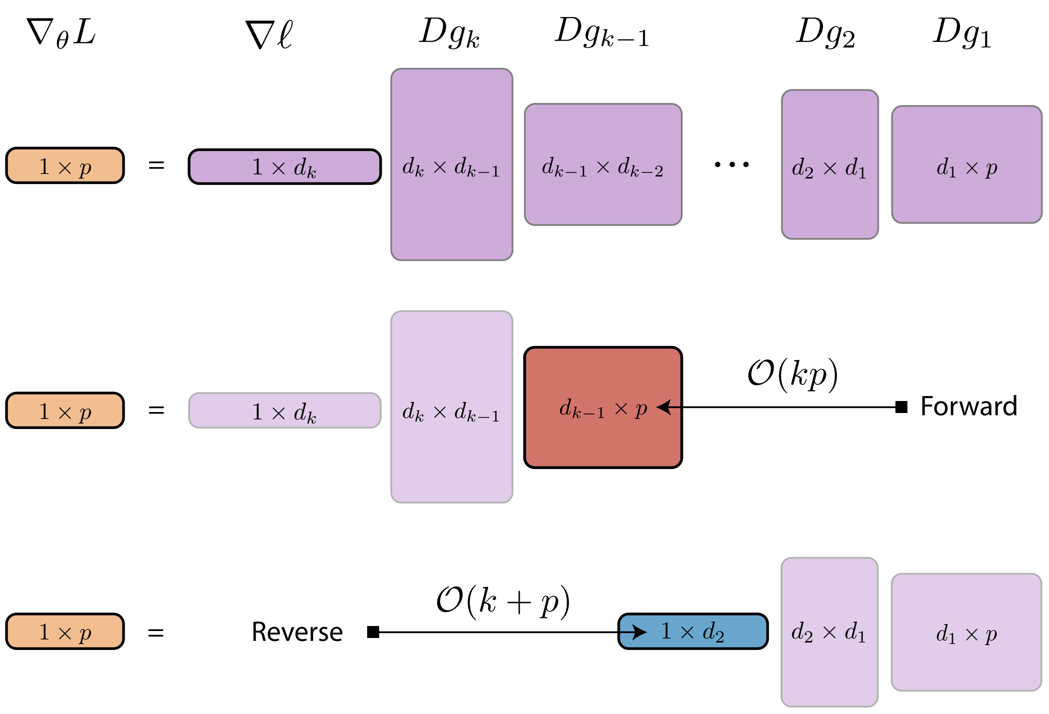


Figure 2.2: *Comparison between forward and reverse AD. Changing the order of Jacobian multiplications changes the total number of floating-point operations, which leads to different computational complexities between forward and reverse mode. When the multiplication is carried from the right side of the mathematical expression for $\nabla_\theta L$, each matrix simplification involves a matrix with size p, giving a total complexity of $\mathcal{O}(kp)$. This is the opposite of what happens when we carried the VJP from the left side of the expression, where the matrix of size $d_1 \times p$ has no effect in the intermediate calculations, making all the intermediate calculations $\mathcal{O}(1)$ with respect to p and a total complexity of $\mathcal{O}(k+p)$.*

where i is the imaginary unit satisfying $i^2 = -1$. The order of this approximation can be found using the Taylor expansion of a function,

$$L(\theta + i\varepsilon) = L(\theta) + i\varepsilon \frac{dL}{d\theta} - \frac{1}{2}\varepsilon^2 \frac{d^2 L}{d\theta^2} + \mathcal{O}(\varepsilon^3). \tag{2.28}$$

Computing the imaginary part $\text{Im}(L(\theta + i\varepsilon))$ leads to

$$\frac{dL}{d\theta} = \frac{\text{Im}(L(\theta + i\varepsilon))}{\varepsilon} + \mathcal{O}(\varepsilon^2). \tag{2.29}$$

The method of *complex step differentiation* consists then in estimating the gradient as $\text{Im}(L(\theta + i\varepsilon))/\varepsilon$ for a small value of ε. Besides the advantage of being a method with precision $\mathcal{O}(\varepsilon^2)$, the complex step method avoids subtracting cancellation error and then the value of ε can be reduced to almost machine precision error without affecting the calculation of the derivative. However, a major limitation of this method is that it only works for complex analytical functions (Martins et al. 2003). Extension to higher order derivatives can be done by introducing multicomplex variables (Lantoine et al. 2012).

2.3.5 Symbolic differentiation

In symbolic differentiation, functions are represented algebraically instead of algorithmically, which is why many symbolic differentiation tools are included inside computer algebra systems (CAS) (Gowda et al. 2022). Instead of numerically evaluating the final value of a derivative, symbolic systems define *algebraic* objects, including variable names, expressions, operations, and literals. For example, the relation $y = x^2$ is interpreted as expression with two variables, x and y, and the symbolic system need to generate the derivative $y' = 2 \times x$ with 2 a numeric literal, $\times$ a binary operation, and x the same variable assignment than in the original expression. When the function to differentiate is large, symbolic differentiation can lead to *expression swell*, that is, exponentially large or complex symbolic expressions (Baydin et al. 2017). Here, an important piece of CAS is simplification routines that reduce the size and complexity of algebraic expressions by finding common sub-expressions. This can make symbolic differentiation very efficient when computing derivatives multiple times and for different input values (Dürrbaum et al. 2002).

It is important to remark on the close relationship between AD and symbolic differentiation. There is no agreement as to whether symbolic differentiation should be classified as AD (Elliott 2018; Juedes 1991; Laue 2019) or as a different method (Baydin et al. 2017). Both are equivalent in the sense that they perform the same operations but the underlying data structure is different (Laue 2019). Here, expression swell is a consequence of the underlying representation when this does not allow for common sub-expressions. This can also be understood as if AD is symbolic differentiation performed by a compiler (Elliott 2018), meaning that different AD can be classified based on the level of integration with the underlying source language (Juedes 1991)

2.3.6 Sensitivity equations

An easy way to derive an expression for the sensitivity s defined in Equation (2.13) is by deriving the sensitivity equations (Ramsay et al. 2017), a method also referred to as continuous local sensitivity analysis (CSA). If we consider the original system of ODEs given by Equation (2.1) and we differentiate with respect to θ, we then obtain

$$\frac{d}{d\theta}\left(\frac{du}{dt} - f(u(\theta), \theta, t)\right) = 0. \tag{2.30}$$

Assuming that an unique solution exists and both $\frac{\partial f}{\partial u}$ and $\frac{\partial f}{\partial \theta}$ are continuous in the neighbourhood of the solution, or under the guarantee of interchangeability of the derivatives

(Gronwall 1919), for example by assuming that both $\frac{du}{dt}$ and $\frac{du}{d\theta}$ are differentiable (Palmieri et al. 2020), we can derive

$$\frac{d}{d\theta}\frac{du}{dt} = \frac{d}{d\theta} f(u(\theta), \theta, t) = \frac{\partial f}{\partial \theta} + \frac{\partial f}{\partial u}\frac{\partial u}{\partial \theta}. \tag{2.31}$$

Identifying the sensitivity matrix $s(t)$, we obtain the *sensitivity differential equation*

$$\frac{ds}{dt} = \frac{\partial f}{\partial u} s + \frac{\partial f}{\partial \theta}. \tag{2.32}$$

Both the original system of n ODEs and the sensitivity equation of np ODEs are solved simultaneously, which is necessary since the sensitivity differential equation directly depends on the value of $u(t)$. This implies that as we solve the ODEs, we can ensure the same level of numerical precision for the two of them inside the numerical solver.

In contrast to the methods previously introduced, the sensitivity equations find the gradient by solving a new set of continuous differential equations. Notice also that the obtained sensitivity $s(t)$ can be evaluated at any given time t. This method can be labeled as forward, since we solve both $u(t)$ and $s(t)$ as we solve the differential equation forward in time, without the need of backtracking any operation though the solver. By solving the sensitivity equation and the original differential equation for $u(t)$ simultaneously, we ensure that by the end of the forward step we have calculated both $u(t)$ and $s(t)$.

2.3.7 Discrete adjoint method

Also known as the adjoint state method, it is another example of a discrete method that aims to find the gradient by solving an alternative system of linear equations, known as the *adjoint equations*, simultaneously with the original system of equations defined by the numerical solver. These methods are extremely popular in optimal control theory in fluid dynamics, for example for the design of geometries for vehicles and airplanes that optimize performance (Elliott et al. 1996; Giles et al. 2000b).

The idea of the adjoint method is to treat the differential equation as a constraint in an optimization problem and then differentiate an objective function subject to that constraint. Mathematically speaking, this can be treated both from a duality or Lagrangian perspective (Giles et al. 2000b). In agreement with other authors, we prefer to derive the equation using the former as it gives better insights to how the method works and it allows generalization to other user cases (Givoli 2021).

2.3.7.1 Adjoint state equations

The derivation of the discrete adjoint equations is carried out once the numerical scheme for solving Equation (2.1) has been specified. Given a discrete sequence of timesteps $t_0, t_1, \dots, t_N$, we aim to find approximate numerical solutions $u_i \approx u(t_i; \theta)$. Any numerical solver, including the ones discussed in Section 2.3.1.1, can be understood as solving the (in general nonlinear) system of equations defined by $G(U; \theta) = 0$, where U is the super-vector

$U = (u_1, u_2, \ldots, u_N) \in \mathbb{R}^{nN}$, and we had combine the systems of equations defined by the iterative solver as $G(U; \theta) = (g_1(u_1; \theta), \ldots, g_N(u_N; \theta)) = 0$ (see Equation (2.4)).

We are interested in differentiating an objective or loss function $L(U, \theta)$ with respect to the parameter θ. Since here U is the discrete set of evaluations of the solver, examples of loss functions now include

$$L(U, \theta) = \frac{1}{2} \sum_{i=1}^{N} \| u_i - u_i^{\text{obs}} \|^2, \tag{2.33}$$

with u_i^{obs} the observed time-series. Now, same as Equation (2.11) we have

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} + \frac{\partial L}{\partial U} \frac{\partial U}{\partial \theta}. \tag{2.34}$$

We further need to impose the constraint that the solution satisfies the algebraic equation $G(U; \theta) = 0$, which gives

$$\frac{dG}{d\theta} = \frac{\partial G}{\partial \theta} + \frac{\partial G}{\partial U} \frac{\partial U}{\partial \theta} = 0 \tag{2.35}$$

and which is equivalent to

$$\frac{\partial U}{\partial \theta} = - \left(\frac{\partial G}{\partial U} \right)^{-1} \frac{\partial G}{\partial \theta}. \tag{2.36}$$

If we replace this last expression into equation (2.34), we obtain

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \frac{\partial L}{\partial U} \left(\frac{\partial G}{\partial U} \right)^{-1} \frac{\partial G}{\partial \theta}. \tag{2.37}$$

The important trick in the adjoint state methods is to observe that in this last equation, the right-hand side can be resolved as a vector-Jacobian product (VJP), with $\frac{\partial L}{\partial U}$ being the vector. Instead of computing the product of the matrices $\left(\frac{\partial G}{\partial U}\right)^{-1}$ and $\frac{\partial G}{\partial \theta}$, it is computationally more efficient first to compute the resulting vector from the VJP operation $\frac{\partial L}{\partial U} \left(\frac{\partial G}{\partial U}\right)^{-1}$ and then multiply this by $\frac{\partial G}{\partial \theta}$. This leads to the definition of the adjoint $\lambda \in \mathbb{R}^{nN}$ as the solution of the linear system of equations

$$\left(\frac{\partial G}{\partial U} \right)^T \lambda = \left(\frac{\partial L}{\partial U} \right)^T, \tag{2.38}$$

or equivalently,

$$\lambda^T = \frac{\partial L}{\partial U} \left(\frac{\partial G}{\partial U} \right)^{-1}. \tag{2.39}$$

Finally, if we replace Equation (2.39) into (2.37), we obtain

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \lambda^T \frac{\partial G}{\partial \theta}. \tag{2.40}$$

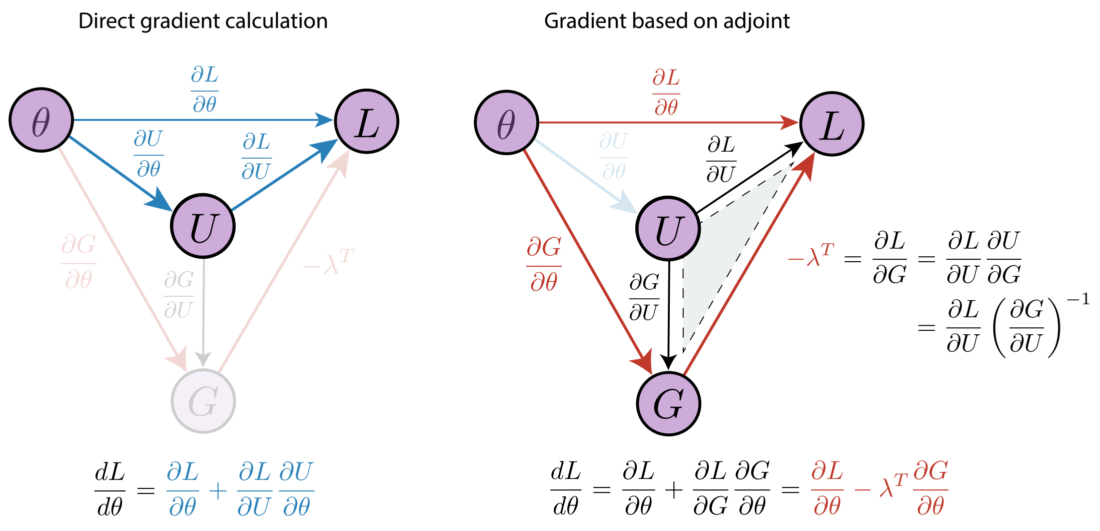

Direct gradient calculation
Gradient based on adjoint
θ
L
U
G
$\frac{\partial L}{\partial \theta}$
$\frac{\partial U}{\partial \theta}$
$\frac{\partial L}{\partial U}$
$\frac{\partial G}{\partial \theta}$
$\frac{\partial G}{\partial U}$
$-\lambda^T$
$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \theta}$
$-\lambda^T = \frac{\partial L}{\partial G} = \frac{\partial L}{\partial U}\frac{\partial U}{\partial G} = \frac{\partial L}{\partial U}\left(\frac{\partial G}{\partial U}\right)^{-1}$
$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} + \frac{\partial L}{\partial G}\frac{\partial G}{\partial \theta} = \frac{\partial L}{\partial \theta} - \lambda^T\frac{\partial G}{\partial \theta}$

Figure 2.3: *Diagram showing how gradients are computed using discrete adjoints. On the left, we see how gradients will be computed if we use the chain rule applied to the directed triangle defined by the variables θ, U, and L (blue arrows). However, we can define the intermediate vector variable $G = G(U;\theta)$, which satisfies $G = 0$ as long as the discrete system of differential equations are satisfied, and apply the chain rule instead to the triangle defined by θ, G, and L (red arrows). In the red diagram, the calculation of $\frac{\partial L}{\partial G}$ is done by pivoting in U as shown in the right diagram (shaded area). Notice that the use of adjoints avoids the calculation of the sensitivity $\frac{\partial U}{\partial \theta}$. The adjoint is defined as the partial derivative $\lambda^T = -\frac{\partial L}{\partial G}$ representing changes in the loss function due to variations in the discrete equation $G(U;\theta) = 0$.*

The important trick used in the adjoint method is the rearrangement of the multiplicative terms involved in equation (2.37). Computing the full Jacobian/sensitivity $\partial U/\partial\theta$ will be computationally expensive and involves the product of two matrices. However, we are not interested in the calculation of the Jacobian, but instead in the VJP given by $\frac{\partial L}{\partial U}\frac{\partial U}{\partial \theta}$. By rearranging these terms and relying in the intermediate variable $G(U;\theta)$, we can make the same computation more efficient. These ideas are summarized in the diagram in Figure 2.3, where we can also see an interesting interpretation of the adjoint as being equivalent to $\lambda^T = -\frac{\partial L}{\partial G}$.

Notice that the algebraic equation of the adjoint λ in Equation (2.38) is a linear system of equations even when the original system $G(U;\theta) = 0$ was not necessarily linear in U. This means that while the forward mode may require multiple iterations in order to solve the non-linear system $G(U) = 0$ (e.g., by using Krylov methods), the backwards step to compute the adjoint is one single linear system of equations.

2.3.7.2 Simple linear system

To gain further intuition about the discrete adjoint method, let us consider the simple case of the explicit linear one-step methods, where at every step we need to solve the equation $u_{i+1} = g_i(u_i; \theta) = A_i(\theta)\, u_i + b_i(\theta)$, where $A_i(\theta) \in \mathbb{R}^{n\times n}$ and $b_i(\theta) \in \mathbb{R}^n$ are defined by the numerical solver (Johnson 2012). This condition can be written in a more compact manner as $G(U) = A(\theta)U - b(\theta) = 0$, that is

$$A(\theta)U = \begin{bmatrix} \mathbb{I}_n & 0 & & & \\ -A_1 & \mathbb{I}_n & 0 & & \\ & -A_2 & \mathbb{I}_n & 0 & \\ & & & \ddots & \\ & & & -A_{N-1} & \mathbb{I}_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} A_0 u_0 + b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{bmatrix} = b(\theta), \tag{2.41}$$

with $\mathbb{I}_n$ the identity matrix of size $n \times n$. Notice that in most cases, the matrix $A(\theta)$ is quite large and mostly sparse. While this representation of the discrete differential equation is convenient for mathematical manipulations, when solving the system we rely on iterative solvers that save memory and computation.

For the linear system of discrete equations $G(U; \theta) = A(\theta)U - b(\theta) = 0$, we have

$$\frac{\partial G}{\partial \theta} = \frac{\partial A}{\partial \theta} U - \frac{\partial b}{\partial \theta}, \tag{2.42}$$

so the desired gradient in Equation (2.40) can be computed as

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \lambda^T \left(\frac{\partial A}{\partial \theta} U - \frac{\partial b}{\partial \theta} \right) \tag{2.43}$$

with λ the discrete adjoint obtained by solving the linear system in Equation (2.38),

$$A(\theta)^T \lambda = \begin{bmatrix} \mathbb{I}_n & -A_1^T & & & \\ 0 & \mathbb{I}_n & -A_2^T & & \\ & 0 & \mathbb{I}_n & -A_3^T & \\ & & & \ddots & -A_{N-1}^T \\ & & & 0 & \mathbb{I}_n \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \vdots \\ \lambda_N \end{bmatrix} = \begin{bmatrix} u_1 - u_1^{\text{obs}} \\ u_2 - u_2^{\text{obs}} \\ u_3 - u_3^{\text{obs}} \\ \vdots \\ u_N - u_N^{\text{obs}} \end{bmatrix} = \frac{\partial L}{\partial U}^T. \tag{2.44}$$

This is a linear system of equations with the same size of the original $A(\theta)U = b(\theta)$, but involving the adjoint matrix A^T. Computationally this also means that if we can solve the original system of discretized equations then we can also solve the adjoint at the same computational cost (e.g., by using the LU factorization of $A(\theta)$). Another more natural way of finding the adjoints λ is by noticing that the system of equations (2.44) is equivalent to the final value problem

$$\lambda_i = A_i^T \lambda_{i+1} + (u_i - u_i^{\text{obs}}) \tag{2.45}$$

with final condition λ_N. This means that we can solve the adjoint equation backwards, i.e., in reverse mode, starting from the final state λ_N and computing the values of λ_i in decreasing index order. Unless the loss function L is linear in U, this procedure requires to know the value of u_i (or some equivalent form of it) at any given timestep.

2.3.8 Continuous adjoint method

The continuous adjoint method, also known as continuous adjoint sensitivity analysis (CASA), operates by defining a convenient set of new differential equations for the adjoint variable and using this to compute the gradient in a more efficient manner. We encourage the interested reader to make the effort of following how the continuous adjoint method follows the same logic as the discrete adjoint method, but where the discretization of the differential equation does not happen until the very last step, when the solutions of the differential equations involved need to be numerically evaluated.

Consider an integrated loss function defined in Equation (2.8) of the form

$$L(u;\theta) = \int_{t_0}^{t_1} h(u(t;\theta),\theta)dt \tag{2.46}$$

and its derivative with respect to the parameter θ given by the following integral involving the sensitivity matrix $s(t)$:

$$\frac{dL}{d\theta} = \int_{t_0}^{t_1} \left(\frac{\partial h}{\partial \theta} + \frac{\partial h}{\partial u} s(t) \right) dt. \tag{2.47}$$

Just as in the case of the discrete adjoint method, the complicated term to evaluate in the last expression is the sensitivity (Equation (2.13)). Again, the trick is to evaluate the VJP $\frac{\partial h}{\partial u}\frac{\partial u}{\partial \theta}$ by first defining an intermediate adjoint variable. The continuous adjoint equation now is obtained by finding the dual/adjoint equation of the sensitivity equation using the weak formulation of Equation (2.32). The adjoint equation is obtained by writing the sensitivity equation in the form

$$\int_{t_0}^{t_1} \lambda(t)^T \left(\frac{ds}{dt} - f(u,\theta,t)\, s - \frac{\partial f}{\partial \theta} \right) dt = 0, \tag{2.48}$$

where this equation must be satisfied for every function $\lambda(t)$ in order for Equation (2.32) to be true. The next step is to get rid of the time derivative applied to the sensitivity $s(t)$ using integration by parts:

$$\int_{t_0}^{t_1} \lambda(t)^T \frac{ds}{dt} dt = \lambda(t_1)^T s(t_1) - \lambda(t_0)^T s(t_0) - \int_{t_0}^{t_1} \frac{d\lambda^T}{dt} s(t)\, dt. \tag{2.49}$$

Replacing this last expression into Equation (2.48) we obtain

$$\int_{t_0}^{t_1} \left(-\frac{d\lambda^T}{dt} - \lambda(t)^T f(u,\theta,t) \right) s(t) dt = \int_{t_0}^{t_1} \lambda(t)^T \frac{\partial f}{\partial \theta} dt - \lambda(t_1)^T s(t_1) + \lambda(t_0)^T s(t_0). \tag{2.50}$$

At first glance, there is nothing particularly interesting about this last equation. However, both Equations (2.47) and (2.50) involve a VJP with $s(t)$. Since Equation (2.50) must hold for every function $\lambda(t)$, we can pick $\lambda(t)$ to make the terms involving $s(t)$ in Equations (2.47) and (2.50) to perfectly coincide. This is done by defining the adjoint $\lambda(t)$ to be the solution of the new system of differential equations

$$\frac{d\lambda}{dt} = -f(u,\theta,t)^T\lambda - \frac{\partial h^T}{\partial u} \qquad \lambda(t_1) = 0. \tag{2.51}$$

Notice that the adjoint equation is defined with the final condition at t_1, meaning that it needs to be solved backwards in time. The definition of the adjoint $\lambda(t)$ as the solution of this last ODE simplifies Equation (2.50) to

$$\int_{t_0}^{t_1} \frac{\partial h}{\partial u} s(t) dt = \lambda(t_0)^T s(t_0) + \int_{t_0}^{t_1} \lambda(t)^T \frac{\partial f}{\partial \theta} dt. \tag{2.52}$$

Finally, replacing this inside the expression for the gradient of the loss function we have

$$\frac{dL}{d\theta} = \lambda(t_0)^T s(t_0) + \int_{t_0}^{t_1} \left(\frac{\partial h}{\partial \theta} + \lambda^T \frac{\partial f}{\partial \theta} \right) dt \tag{2.53}$$

The full algorithm to compute the full gradient $\frac{dL}{d\theta}$ can be described as follows:

1. Solve the original differential equation $\frac{du}{dt} = f(u,t,\theta)$;
2. Solve the backwards adjoint differential equation given by Equation (2.51);
3. Compute the gradient using Equation (2.53).

The same recipe used here to derived the continuous adjoint method for a system of ODEs can be employed to derive adjoint methods for PDEs (Giles et al. 2000b) and DAE (Cao et al. 2002).

2.3.9 Mathematical comparison of the methods

In Sections 2.3.2-2.3.8 we focused in merely introducing each one of the sensitivity methods classified in Figure 2.1 as separate methods, postponing the discussion about their common points. In this section, we are going to compare these methods one-to-one to show parallelism across them. We hope this enlightens the discussion on sensitivity methods and helps demystify misconceptions around the sometimes apparent differences across methods. This comparison will be strengthen again later in Section 2.4, where we will see how even small differences between methods can be translated to different software implementations with different advantages.

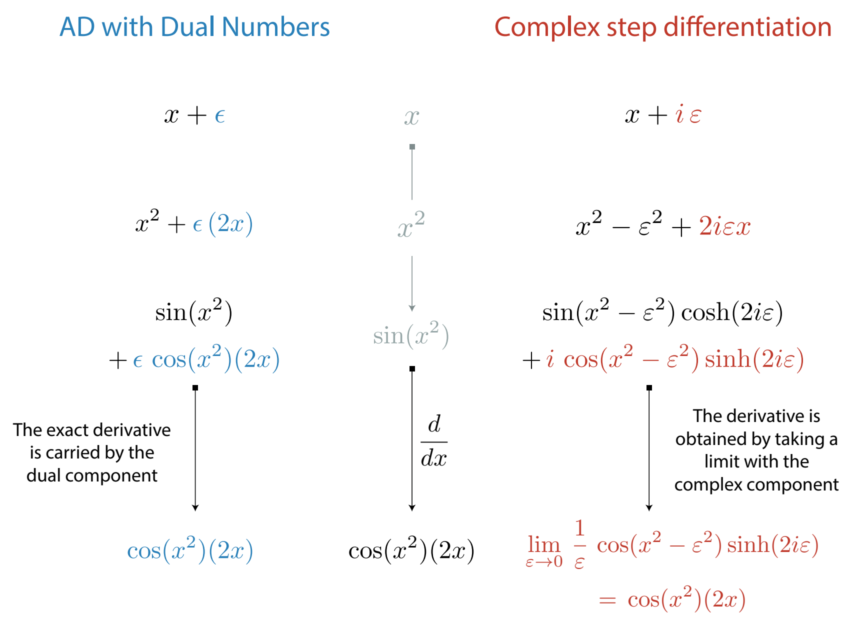

AD with Dual Numbers
Complex step differentiation
$x + \epsilon$
x
$x + i\varepsilon$
$x^2 + \epsilon\,(2x)$
x^2
$x^2 - \varepsilon^2 + 2i\varepsilon x$
$\sin(x^2) + \epsilon\,\cos(x^2)(2x)$
$\sin(x^2)$
$\sin(x^2-\varepsilon^2)\cosh(2i\varepsilon) + i\,\cos(x^2-\varepsilon^2)\sinh(2i\varepsilon)$
The exact derivative is carried by the dual component
$\frac{d}{dx}$
The derivative is obtained by taking a limit with the complex component
$\cos(x^2)(2x)$
$\cos(x^2)(2x)$
$\lim_{\varepsilon\to 0}\frac{1}{\varepsilon}\cos(x^2-\varepsilon^2)\sinh(2i\varepsilon) = \cos(x^2)(2x)$

Figure 2.4: *Comparison between AD implemented with dual numbers and complex step differentiation. For the simple case of the function $f(x) = \sin(x^2)$, we can see how each operation is carried in the forward step by the dual component (blue) and the complex component (red). Whereas AD gives the exact gradient at the end of the forward run, in the case of the complex step method we need to take the limit in the imaginary part.*

2.3.9.1 Forward AD and complex step differentiation

Notice that both AD based on dual number and complex-step differentiation introduce an abstract unit (ϵ and i, respectively) associated with the imaginary part of the dual variable that carries forward the numerical value of the gradient. Although these methods seem similar, we emphasize that AD gives the exact gradient, whereas complex step differentiation relies on numerical approximations that are valid only when the stepsize ε is small. In Figure 2.4 we show how the calculation of the gradient of the function $\sin(x^2)$ is performed by these two methods. Whereas the second component of the dual number has the exact derivative of the function $\sin(x^2)$ with respect to x, it is not until we take $\varepsilon \to 0$ that we obtain the derivative in the imaginary component for the complex step method. The dependence of the complex step differentiation method on the step size gives it a closer resemblance to finite difference methods than to AD using dual numbers. Furthermore, the complex step method involves more terms in the calculation, a consequence of the fact that second order terms of the form $i^2 = -1$ are transferred to the real part of the complex number, while for dual numbers the terms associated to $\epsilon^2 = 0$ vanish (Martins et al. 2001).

2.3.9.2 Discrete adjoints and reverse AD

Both discrete adjoint methods and reverse AD are classified as discrete and reverse methods (see Figure 2.1). Furthermore, both methods introduce an intermediate adjoint associated to the partial derivative of the loss function (output variable) with respect to intermediate variables of the forward computation. In the case of reverse AD, the adjoint is defined with the notation $\bar{w}$ (Equation (2.23)), while in the discrete adjoint method this correspond to each one of the variables $\lambda_1, \lambda_2, \ldots, \lambda_N$ (Equation (2.44)). In this section we will show that both methods are mathematically equivalent (Li et al. 2020a; Zhu et al. 2021b), but naive implementations using reverse AD can result in sub-optimal performances compared to the one obtained by directly employing the discrete adjoint method (Alexe et al. 2009).

In order to have a better idea of how this works in the case of a numerical solver, let us consider again the case of a one-step explicit method, not necessarily linear, where the updates u_i satisfy the equation $u_{i+1} = g_i(u_i, \theta)$. Following the same schematics than in Figure 2.3, we represent the computational graph of the numerical method in Figure 2.5 using the intermediate variables $g_1, g_2, \ldots, g_{N-1}$. The dual/adjoint variables defined in reverse AD in this computational graph are given by

$$\bar{g}_i = \frac{\partial u_{i+1}}{\partial g_i} \bar{u}_{i+1} = \frac{\partial L}{\partial u_{i+1}} + \frac{\partial g_{i+1}}{\partial u_i} \bar{g}_{i+1}. \tag{2.54}$$

The updates of $\bar{g}_i$ then mathematically coincide with the updates in reverse mode of the adjoint variable λ_i (see Equation (2.45)).

Modern numerical solvers use functions g_i that correspond to nested functions, meaning $g_i = g_i^{(k_i)} \circ g_i^{(k_i - 1)} \circ \ldots \circ g_i^{(1)}$. This is certainly the case for implicit methods when u_{i+1} is the solution of an iterative Newton's method (Hindmarsh et al. 2005); or in cases where the numerical solver includes internal iterative sub-routines (Alexe et al. 2009). If the number of intermediate function is large, reverse AD will result in a large computational graph, potentially leading to excessive memory usage and slow computation (Alexe et al. 2009; Margossian 2018). A solution to this problem is to introduce a customized *super node* that directly encapsulates the contribution to the full adjoint in g_i without computing the adjoint for each intermediate function $g_i^{(j)}$. Provided with the value of the Jacobian matrices $\frac{\partial g_i}{\partial u_i}$ and $\frac{\partial g_i}{\partial \theta}$, we can use the implicit function theorem to find the $\frac{\partial u_i}{\partial \theta}$ as the solution of the linear system of equations

$$\frac{\partial g_i}{\partial u_i} \frac{\partial u_i}{\partial \theta} = - \frac{\partial g_i}{\partial \theta} \tag{2.55}$$

and implement AD by directly solving this new system of equations. In both cases, the discrete adjoint method can be implemented directly on top of a reverse AD tool that allows customized adjoint calculation (Rackauckas et al. 2021). Furthermore, notice that instead of the full Jacobian, reverse methods only required to compute VJPs.

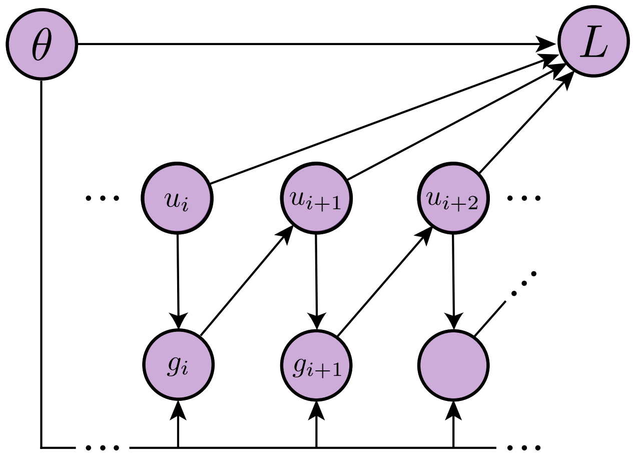


Figure 2.5: *Computational graph associated to the discrete adjoint method. Reverse AD applied on top of the computational graph leads to the update rules for the discrete adjoint. The adjoint variable λ_i in the discrete adjoint method coincides with the adjoint variable $\bar{g}_i$ defined in the backpropagation step.*

2.3.9.3 Consistency: forward AD and sensitivity equations

The sensitivity equations can also be solved in discrete forward mode by numerically discretizing the original ODE and later deriving the discrete sensitivity equations (Ma et al. 2021a). For most cases, this leads to the same result than in the continuous case (Zhang et al. 2014). We can numerically solve for the sensitivity s by extending the parameter θ to a multidimensional dual number

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{bmatrix} \longrightarrow \begin{bmatrix} \theta_1 + \epsilon_1 \\ \theta_2 + \epsilon_2 \\ \vdots \\ \theta_p + \epsilon_p \end{bmatrix} \tag{2.56}$$

where $\epsilon_i \epsilon_j = 0$ for all pairs of i and j (see Section 2.3.3.1.1). The dependency of the solution u of the ODE on the parameter θ is now expanded following Equation (2.20) as

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \longrightarrow \begin{bmatrix} u_1 + \sum_{j=1}^p \frac{\partial u_1}{\partial \theta_j} \epsilon_j \\ u_2 + \sum_{j=1}^p \frac{\partial u_2}{\partial \theta_j} \epsilon_j \\ \vdots \\ u_p + \sum_{j=1}^p \frac{\partial u_n}{\partial \theta_j} \epsilon_j \end{bmatrix} = u \, + \, s \, \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_p \end{bmatrix}, \tag{2.57}$$

that is, the dual component of the vector u corresponds exactly to the sensitivity matrix s. This implies forward AD applied to any multistep linear solver will result in the application of the same solver to the sensitivity equation (Equation (2.32)). For example, for the forward Euler method this gives

$$\begin{aligned} u^{t+1} + s^{t+1}\,\epsilon &= u^t + s^t\,\epsilon + \Delta t\, f(u^t + s^t\,\epsilon, \theta + \epsilon, t) \\ &= u^t + f(u^t, \theta, t) + \Delta t \left(\frac{\partial f}{\partial u} s^t + \frac{\partial f}{\partial \theta} \right) \epsilon. \end{aligned} \tag{2.58}$$

The dual component corresponds to the forward Euler discretization of the sensitivity equation, with s^t the temporal discretization of the sensitivity $s(t)$.

The consistency result for discrete and continuous methods also holds for Runge-Kutta methods (Walther 2007). When introducing dual numbers, the Runge-Kutta scheme in Equation (2.2) gives the following identities

$$u^{n+1} + s^{n+1}\epsilon = s_n + \Delta t_n \sum_{i=1}^{s} b_i \dot{k}_i \tag{2.59}$$

$$k_i + \dot{k}_i \epsilon = f\left(u^n + \sum_{j=1}^{s} a_{ij} k_j + \left(s^n + \sum_{j=1}^{s} a_{ij} \dot{k}_j \right) \epsilon, \theta + \epsilon, t_n + c_i \Delta t_n \right) \tag{2.60}$$

with $\dot{k}_i$ the dual variable associated to k_i. The partial component in Equation (2.60) carrying the coefficient ϵ gives

$$\begin{aligned} \dot{k}_i = &\frac{\partial f}{\partial u}\left(u^n + \sum_{j=1}^{s} a_{ij} k_j, \theta, t_n + c_i \Delta t_n \right) \left(s^n + \sum_{j=1}^{s} a_{ij} \dot{k}_j \right) \\ &+ \frac{\partial f}{\partial \theta}\left(u^n + \sum_{j=1}^{s} a_{ij} k_j, \theta, t_n + c_i \Delta t_n \right), \end{aligned} \tag{2.61}$$

which coincides with the Runge-Kutta scheme we will obtain for the original sensitivity equation. This means that forward AD on Runge-Kutta methods leads to solutions for the sensitivity that have the same convergence properties of the forward solver.

2.3.9.4 Consistency: discrete and continuous adjoints

As previously mentioned, the difference between the discrete and continuous adjoint methods is that the former follows the discretize-then-differentiate approach, also known as finite difference of adjoints (Sirkes et al. 1997). In contrast, continuous adjoint equations are derived by directly operating on the differential equation, without a priori consideration of the numerical scheme used to solve it. In some sense then, we can think of the discrete adjoint $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$ in Equation (2.44) as the discretization of the continuous adjoint $\lambda(t)$.

A natural question then is if these two methods effectively compute the same gradient, i.e., if the discrete adjoint consistently approximate its continuous counterpart. Furthermore, since the continuous adjoint method requires to numerical solve the adjoint, we are interested in the relative accuracy of the forward and backwards step. It has been shown that for both explicit and implicit Runge-Kutta methods, as long as the coefficients in the numerical scheme given in Equation (2.2) satisfy the condition $b_i \neq 0$ for all $i = 1, 2, \ldots, s$, then the discrete adjoint is a consistent estimate of the continunous adjoint with same order of convergence than the forward numerical solver (Hager 2000; Sandu 2006, 2011; Walther 2007). To guarantee the same order of convergence, it is important that both the forward and backwards solver use the same Runge-Kutta coefficients (Alexe et al. 2009). Importantly, even when consistent, the code generated using the discrete adjoint using AD tools (see Section 2.3.9.2) can be sub-optimal and manual modification of the differentiation code are required to guarantee correctness (Alexe et al. 2007; Eberhard et al. 1996).

It is important to remark that adjoint methods can fail in chaotic systems (Wang et al. 2014). Some works have shown that continuous adjoints can lead to unstable sensitivities (Jensen et al. 2014). In the more general case, discrete and continuous adjoint methods can give different computational results (Sirkes et al. 1997).

2.4 Implementation: A computer science perspective

In this section, we address how these different methods are computationally implemented and how to decide which method to use depending on the scientific task. In order to address this point, it is important to make one further distinction between the methods introduced in Section 2.3, i.e., between those that apply direct differentiation at the algorithmic level and those that are based on numerical solvers. The former are easier to implement since they are agnostic with respect to the mathematical and numerical properties of the system of ODEs; however, they tend to be either inaccurate, memory-expensive, or at times unfeasible for large models. The latter family of methods that are based on numerical solvers include the sensitivity equations and the adjoint methods, both discrete and continuous; they are more difficult to implement and for real case applications require complex software implementations, but they are also more accurate and adequate. This section is then divided in two parts:

- **Direct methods.** (Section 2.4.1) Their implementation occurs at a higher hierarchy than the numerical solver software. They include finite differences, AD, complex step differentiation.
- **Solver-based methods.** Their implementation occurs at the same level of the numerical solver. They include:
  - Sensitivity equations (Section 2.4.2.1)
  - Adjoint methods, both discrete and continuous (Section 2.4.2.2)

Despite the fact that these methods can be implemented in different programming languages, here we decided to use the Julia programming language for the different examples. Julia is a recent but mature programming language that already has a large tradition in implementing packages aiming to advance differentiable programming (Bezanson et al. 2017, 2012), with a strong emphasis in differential equation solvers (Rackauckas et al. 2016) and sensitivity analysis (Rackauckas et al. 2020). Nevertheless, in reviewing existing work, we will also point to applications developed in other programming languages.

The GitHub repository `DiffEqSensitivity-Review` contains both text and code used to generate this manuscript. See Appendix A for a complete description of the scripts provided. The symbol ♣ will be use to reference code associated with scripts in the repository.

2.4.1 Direct methods

Direct methods are implemented independent of the structure of the differential equation and the numerical solver used to solve it. These include finite differences, complex step differentiation, and both forward and reverse mode AD.

2.4.1.1 Finite differences

Finite differences are easy to implement manually, do not require much software support, and provide a direct way of approximating a gradient. In Julia, these methods are implemented in `FiniteDiff.jl` and `FiniteDifferences.jl`, which already include subroutines to determine step-sizes. However, finite differences are less accurate and as costly as forward AD (Griewank 1989) and complex-step differentiation. Figure 2.6 illustrates the error in computing the gradient of a simple loss function for both true analytical solution and numerical solution of a system of ODEs as a function of the stepsize ε using finite differences ♣$_1$. Here we consider the solution of the differential equation $u'' + \omega^2 u = 0$ with initial condition $u(0) = 0$ and $u'(0) = 1$, which has analytical solution $u_{\text{true}}(t) = \sin(\omega t)/\omega$. The numerical solution $u_{\text{num}}(t)$ can be founded by solving the system of ODEs

$$\begin{cases} \frac{du_1}{dt} = u_2 & u_1(0) = 0 \\ \frac{du_2}{dt} = -\omega^2 u_1 & u_2(0) = 1. \end{cases} \tag{2.62}$$

The loss function used to differentiate is given by $L(\theta) = u(10)$. We see that finite differences are inaccurate for computing the derivative of u_{true} with respect to ω (that is, $u'_{\text{true}} = \cos(\omega t) - \sin(\omega t)/\omega^2$) when the stepsize ε is both too small and too large (red dashed line). When the derivative is instead computed using the numerical solution $u_{\text{num}}(t)$ (red circles), the accuracy of the derivative further deteriorates due to approximation errors in the solver. This effect is dependent on the numerical solver tolerance. Notice that in general we do not know the true solution of the differential equation, so the derivative of u_{true} obtained using finite differences just serves as a lower bound of the error we expect to see when performing sensitivity analysis on top of the numerical solver.

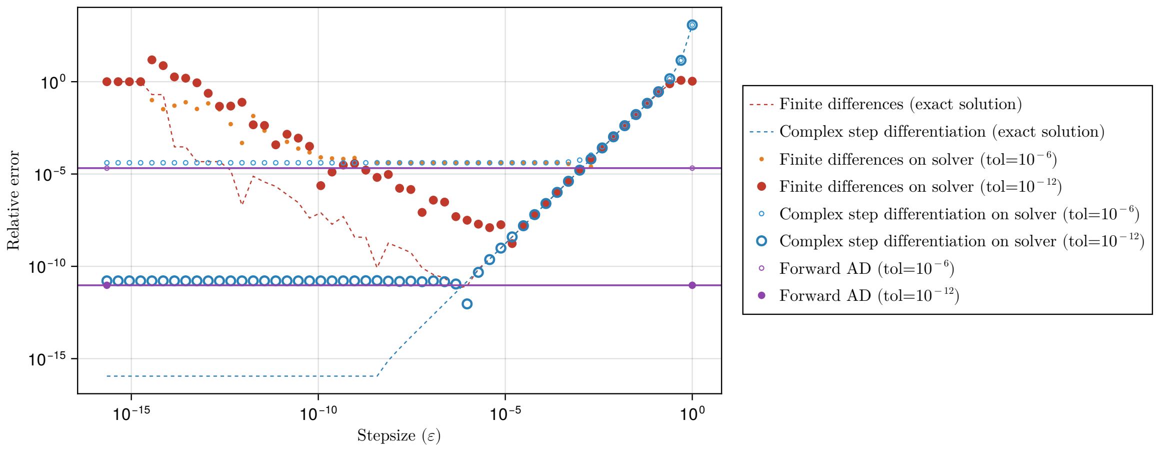


Figure 2.6: *Absolute relative error when computing the gradient of the function $u(t) = \sin(\omega t)/\omega$ with respect to ω at $t = 10.0$ as a function of the stepsize ε. Here, $u(t)$ corresponds to the solution of the differential equation $u'' + \omega^2 u = 0$ with initial condition $u(0) = 0$ and $u'(0) = 1$. The blue dots correspond to the case where the relative error is computed with finite differences. The red and orange lines are for the case where $u(t)$ is numerically computed using the default Tsitouras solver (Tsitouras 2011) from `OrdinaryDiffEq.jl` using different tolerances. The error when using a numerical solver is larger and it is dependent on the numerical precision of the numerical solver.*

2.4.1.2 Automatic differentiation

The AD algorithms described in Section 2.3.3 can be implemented using *operator overloading* for AD based on dual numbers and *source code transformation* for both forward and reverse AD based on the computational graph (Martins et al. 2001). In this section we are going to cover how forward AD is implemented using dual numbers, postponing the implementation of AD based on computational graphs for reverse AD (Section 2.4.1.2.2).

2.4.1.2.1 Forward AD based on dual numbers Implementing forward AD using dual numbers is usually carried out using *operator overloading* (Neuenhofen 2018). This means expanding the object associated to a numerical variable to include the tangent and extending the definition of atomic algebraic functions. In Julia, this can be done by relying on multiple dispatch (Bezanson et al. 2017). The following example illustrates how to define a dual number and its associated binary addition and multiplication extensions ♣$_2$.

```julia
using Base: @kwdef

@kwdef struct DualNumber{F <: AbstractFloat}
    value::F
    derivative::F
end
```

```
# Binary sum
Base.:(+)(a::DualNumber, b::DualNumber) = DualNumber(value = a.value + b.value,
    derivative = a.derivative + b.derivative)

# Binary product
Base.:(*)(a::DualNumber, b::DualNumber) = DualNumber(value = a.value * b.value,
    derivative = a.value*b.derivative + a.derivative*b.value)
```

We further overload base operations for this new type to extend the definition of standard functions by simply applying the chain rule and storing the derivative in the dual variable following Equation (2.20):

```
function Base.:(sin)(a::DualNumber)
    value = sin(a.value)
    derivative = a.derivative * cos(a.value)
    return DualNumber(value=value, derivative=derivative)
end
```

In the Julia ecosystem, `ForwardDiff.jl` implements forward mode AD with multidimensional dual numbers (Revels et al. 2016) and the sensitivity method `ForwardDiff Sensitivity` implements forward differentiation inside the numerical solver using dual numbers. Figure 2.6 shows the result of performing forward AD inside the numerical solver. We can see that for this simple example forward AD performs as good as the best output of finite differences and complex step differentiation (see Section 2.4.1.3) even for the best possible choice of the stepsize ε. Further, note that AD is not subject to numerical errors due to floating point arithmetic (Griewank et al. 2008).

Implementations of forward AD using dual numbers and computational graphs require a number of operations that increases with the number of variables to differentiate, since each computed quantity is accompanied by the corresponding gradient calculations (Griewank 1989). This consideration also applies to the other forward methods, including finite differences and complex-step differentiation, which makes forward models prone to the curse of dimensionality with respect to the number of parameters considered.

Although AD is always algorithmically correct, when combined with a numerical solver *AD can be numerically incorrect* and result in wrong gradient calculations (Eberhard et al. 1996). To illustrate this point, consider the following example of a simple system of ODEs

$$\begin{cases} \frac{du_1}{dt} = au_1 - u_1u_2 & u_1(0) = 1 \\ \frac{du_2}{dt} = -au_2 + u_1u_2 & u_2(0) = 1 \end{cases} \tag{2.63}$$

with a the parameter with respect to which we want to differentiate. In the simple case of $a = 1$, the solutions of the ODE are constant functions $u_1(t) \equiv u_2(t) \equiv 1$. Traditional adaptive stepsize solvers used for just solving ODEs are designed to control for numerical errors in the ODE solution but not in its sensitivities when coupled with an internal AD method. This can lead to wrong gradient calculations that propagate through the numerical solver without further warning $\clubsuit_3$.

2.4.1.2.2 Reverse AD based on computational graph In contrast to finite differences, forward AD, and complex-step differentiation, reverse AD is the only of this family of methods that propagates the gradient in reverse mode by relying on analytical derivatives of primitive functions, which in Julia are available via `ChainRules.jl`. Since this requires the evaluation intermediate variables, reverse AD requires a more delicate protocol of how to store intermediate variables in memory and make them accessible during the backwards pass.

Reverse AD can be implemented via *pullback* functions (Innes 2018), a method also known as *continuation-passing style* (Wang et al. 2019). In the backward step, it executes a series of function calls, one for each elementary operation. If one of the nodes in the graph w is the output of an operation involving the nodes $v_1, \ldots, v_m$, where $v_i \to w$ are all nodes in the graph, then the pullback $\bar{v}_1, \ldots, \bar{v}_m = \mathcal{B}_w(\bar{w})$ is a function that accepts gradients with respect to w (defined as $\bar{w}$) and returns gradients with respect to each v_i (defined as $\bar{v}_i$) by applying the chain rule. Consider the example of the multiplicative operation $w = v_1 \times v_2$. Then

$$\bar{v}_1, \bar{v}_2 \;=\; v_2 \times \bar{w}, v_1 \times \bar{w} \;=\; \mathcal{B}_w(\bar{w}), \tag{2.64}$$

which is equivalent to using the chain rule as

$$\frac{\partial \ell}{\partial v_1} = \frac{\partial}{\partial v_1}(v_1 \times v_2)\frac{\partial \ell}{\partial w}. \tag{2.65}$$

A crucial distinction between AD implementations based on computational graphs is between *static* and *dynamical* methods (Baydin et al. 2017). In the case of a static implementation, the computational graph is constructed before any code is executed, which are encoded and optimized for performance within the graph language. For static structures such as neural networks, this is ideal (Abadi et al. 2016). However, two major drawbacks of static methods are composability with existing code, including support of custom types, and adaptive control flow, which is a common feature of numerical solvers. These issues are addressed in reverse AD implementations using *tracing*, where the program structure is transformed into a list of pullback functions that built the graph dynamically at runtime. Popular libraries in this category are `Tracker.jl` and `ReverseDiff.jl`. There also exist source-to-source AD system that achive highest performance at the same time they support arbitrary control flow structure. These include `Zygote.jl` (Innes et al. 2019), `Enzyme.jl` (Moses et al. 2020), and `Difractor.jl`. The existence of multiple AD packages lead to the development of `AbstractDifferentiation.jl` which allows to combine different methods (Schäfer et al. 2021).

2.4.1.2.3 Checkpointing In opposition to forward methods, all reverse methods, including backpropagation and adjoint methods, require access to the value of intermediate variables during the propagation of the gradient. For a numerical solver, the amount of memory required to accomplish this can be very large, involving a total of at least $\mathcal{O}(nk)$ terms, with k the number of steps of the numerical solver. Checkpointing is a technique that can be used for all the backwards methods that avoids storing all the intermediate states by balancing storing and recomputation to recover the required state exactly (Griewank et al. 2008). This is achieved by saving intermediate states of the solution in the forward pass and recalculating the solution between intermediate states in the backwards mode (Griewank et al. 2008; Schanen et al. 2023).

2.4.1.3 Complex step differentiation

Modern software already has support for complex number arithmetic, making complex step differentiation very easy to implement. In Julia, complex analysis arithmetic can be easily carried inside the numerical solver. The following example shows how to extend the numerical solver used to solve the ODEs in Equation (2.62) to support complex numbers ♣$_4$.

```julia
function dyn!(du::Array{Complex{Float64}}, u::Array{Complex{Float64}}, p, t)
    ω = p[1]
    du[1] = u[2]
    du[2] = - ω^2 * u[1]
end

tspan = [0.0, 10.0]
du = Array{Complex{Float64}}([0.0])
u0 = Array{Complex{Float64}}([0.0, 1.0])

function complexstep_differentiation(f::Function, p::Float64, ε::Float64)
    p_complex = p + ε * im
    return imag(f(p_complex)) / ε
end

complexstep_differentiation(x -> solve(ODEProblem(dyn!, u0, tspan, [x]), Tsit5()
    ).u[end][1], 20., 1e-3)
```

Figure 2.6 further shows the result of performing complex step differentiation using the same example as in Section 2.4.1.1. We can see from both exact and numerical solutions that complex-step differentiation does not suffer from small values of ε, meaning that ε can be chosen arbitrarily small (Martins et al. 2001) as long as it does not reach the underflow threshold (Goldberg 1991). Notice that for large values of the stepsize ε complex step differentiation gives similar results than finite differences, while for small values of ε the performance of complex step differentiation is slightly worse than forward AD. This result emphasizes the observation made in Section 2.3.9.2, complex step differentiation has many aspects in common with finite differences and AD based on dual numbers.

However, the difference between the methods sometimes makes the complex step differentiation more efficient than both finite differences and AD (Lantoine et al. 2012), an effect that can be counterbalanced by the number of extra unnecessary operations that complex arithmetic requires (see last column in Figure 2.4) (Martins et al. 2003).

2.4.2 Solver-based methods

Sensitivity methods based on numerical solvers tend to be better adapted to the structure and properties of the underlying ODE (stiffness, stability, accuracy) but are also more difficult to implement. This difficulty arises from the fact that the sensitivity method needs to deal with some numerical and computational considerations, including i) how to handle matrix/Jacobian-vector products; ii) numerical stability of the forward/reverse solver; and iii) memory-time tradeoff. These factors are further exacerbated by the number of ODEs and parameters in the model. Just a few modern scientific softwares have the capabilities of solving systems of ODEs and computing their sensitivities at the same time. These include `CVODES` within `SUNDIALS` in C (Hindmarsh et al. 2005; Serban et al. 2005); `ODESSA` (Leis et al. 1988) and `FATODE` (discrete adjoints) (Zhang et al. 2014) both in Fortram; `SciMLSensitivity.jl` in Julia (Rackauckas et al. 2020); `Dolfin-adjoint` based on the `FEniCS` Project (Farrell et al. 2013; Mitusch et al. 2019); `DENSERKS` in Fortram (Alexe et al. 2007); `DASPKADJOINT` (Cao et al. 2002); and `Diffrax` in Python (Kidger 2021).

It is important to remark that the underlying machinery of all numerical integrators relies on solvers for linear systems of equations, which can be solved in dense, band (sparse), and Krylov mode. Another important consideration is that all these methods have subroutines to compute the VJPs involved in the sensitivity and adjoint equations. This calculation is carried out by another sensitivity method, usually finite differences or AD, which plays a central role when analyzing the accuracy and stability of the adjoint method.

2.4.2.1 Sensitivity equation

For systems of equations with few numbers of parameters, this method is useful since the system of $n(p+1)$ equations composed by Equations (2.1) and (2.32) can be solved using the same precision for both solution and sensitivity numerical evaluation. Furthermore, it does not required saving the solution in memory. The following example illustrates how Equation (2.62) and the sensitivity equation can be solved simultaneously using the simple explicit Euler method ♣$_5$:

```
ω = 0.2
p = [ω]
u0 = [0.0, 1.0]
tspan = [0.0, 10.0]

# Dynamics
function f(u, p, t)
    du₁ = u[2]
    du₂ = - p[1]^2 * u[1]
    return [du₁, du₂]
end

# Jacobian ∂f/∂p
function ∂f∂p(u, p, t)
    Jac = zeros(length(u), length(p))
    Jac[2,1] = -2*p[1]*u[1]
    return Jac
```

```julia
end

# Jacobian ∂f/∂u
function ∂f∂u(u, p, t)
    Jac = zeros(length(u), length(u))
    Jac[1,2] = 1
    Jac[2,1] = -p[1]^2
    return Jac
end

# Explicit Euler method
function sensitivityequation(u0, tspan, p, dt)
    u = u0
    sensitivity = zeros(length(u), length(p))
    for ti in tspan[1]:dt:tspan[2]
        sensitivity += dt * (∂f∂u(u, p, ti) * sensitivity + ∂f∂p(u, p, ti))
        u += dt * f(u, p, ti)
    end
    return u, sensitivity
end

u, s = sensitivityequation(u0, tspan , p, 0.001)
```

The simplicity of the sensitivity method makes it available in most software for sensitivity analysis. In Julia, the `ForwardSensitivity` method implements continuous sensitivity analysis, which performs forward AD on the solver via `ForwardDiff.jl` (see Section 2.3.9.3). The same result can be achieve by:

```julia
using Zygote, SciMLSensitivity

function f!(du, u, p, t)
    du[1] = u[2]
    du[2] = - p[1]^2 * u[1]
end

s = Zygote.jacobian(p->solve(ODEProblem(f!, u0, tspan, p), Tsit5(), u0=u0, p=p,
     sensealg=ForwardSensitivity())[end], p)
```

Notice that in the last example we needed to re-define the out-of-place function `f` to the in-place version `f!`.

For stiff systems of ODEs the use of the sensitivity equations is unfeasible (Kim et al. 2021). This is because stiff ODEs require the use of stable solvers with cubic cost with respect to the number of ODEs (Wanner et al. 1996), making the total complexity of the sensitivity method $\mathcal{O}(n^3p^3)$. This complexity makes this method useless for models with many ODEs and/or parameters.

2.4.2.1.1 Computing VJPs inside the solver All solver-based methods require the computation of VJPs. In the case of the sensitivity equation, this corresponds to the row/-column terms in $\frac{\partial f}{\partial u}s$ in Equation (2.32). For the adjoint equations, although an efficient trick has been used to remove the computationally expensive VJP, we still need to evaluate the term $\lambda^T\frac{\partial G}{\partial \theta}$ for the discrete adjoint method in Equation (2.39), and $\lambda^T\frac{\partial f}{\partial \theta}$ for the continuous adjoint method in Equation (2.53). Therefore, the choice of the algorithm to compute VJPs can have a significant impact in the overall performance.

In SUNDIALS, the VJPs involved in the sensitivity and adjoint method are handled using finite differences unless specified by the user (Hindmarsh et al. 2005). In FATODE, these can be computed with finite differences, AD, or it can be provided by the user. In the Julia ecosystem, different AD packages are available for this task (see Section 2.4.1.2.2), including ForwardDiff.jl, ReverseDiff.jl, Zygote.jl (Innes et al. 2019), Enzyme.jl (Moses et al. 2020), Tracker.jl. These will compute the VJPs using some form of AD, which will result in correct calculations but potentially sub-optimal code. In Julia, the options autodiff and autojacvec allow to customized if VJPs are computed using AD or finite differences.

2.4.2.2 Adjoint methods

For complex and large systems, direct methods for computing the gradient on top of the numerical solver can be memory expensive due to the large number of function evaluations required by the solver and the later store of the intermediate states. For these cases, adjoint-based methods allow us to compute the gradients of a loss function by instead computing the adjoint that serves as a bridge between the solution of the ODE and the final gradient. The adjoint method offers advantages when working with complex systems. Since we are dealing with a new differential equation special care needs to be taken with respect to numerical efficiency and stability.

2.4.2.2.1 Discrete adjoint method In order to illustrate how the discrete adjoint method works, the following example shows how to manually solve for the gradient of the solution of (2.62) using an explicit Euler method.

```
function discrete_adjoint_method(u0, tspan, p, dt)
    u = u0
    times = tspan[1]:dt:tspan[2]

    λ = [1.0, 0.0]
    ∂L∂θ = zeros(length(p))
    u_store = [u]

    # Forward pass to compute solution
    for t in times[1:end-1]
        u += dt * f(u, p, t)
        push!(u_store, u)
    end

    # Reverse pass to compute adjoint
    for (i, t) in enumerate(reverse(times)[2:end])
        u_memory = u_store[end-i+1]
        λ += dt * ∂f∂u(u_memory, p, t)' * λ
        ∂L∂θ += dt * λ' * ∂f∂p(u_memory, p, t)
    end

    return ∂L∂θ
```

	Method	Stability	Stiff Performance	Memory
Discrete	ReverseDiffAdjoint	Best	$\mathcal{O}(n^3+p)$	High
	ZygoteAdjoint	Best	$\mathcal{O}(n^3+p)$	High
	TrackerAdjoint	Best	$\mathcal{O}(n^3+p)$	High
Continuous	Sensitivity equation	Good	$\mathcal{O}(n^3p^3)$	$\mathcal{O}(1)$
	Backsolve adjoint	Poor	$\mathcal{O}((n+p)^3)$	$\mathcal{O}(1)$
	Backsolve adjoint◀	Medium	$\mathcal{O}((n+p)^3)$	$\mathcal{O}(k)$
	Interpolating adjoint	Good	$\mathcal{O}((n+p)^3)$	High
	Interpolating adjoint◀	Good	$\mathcal{O}((n+p)^3)$	$\mathcal{O}(k)$
	Quadrature adjoint	Good	$\mathcal{O}(n^3+p)$	High
	Gauss adjoint	Good	$\mathcal{O}(n^3+p)$	High

Table 2.1: *Comparison in performance and cost of solver-based methods. Methods that are being checkpointed are indicated with the symbol ◀, with k the total number of checkpoints.*

```julia
end

∂L∂θ = discrete_adjoint_method(u0, tspan, p, 0.001)
```

In this case, the full solution in the forward pass is stored in memory and used to compute the adjoint and integrate the loss function during the reverse pass. As in the case of reverse AD, checkpointing can be used here.

The previous example shows a manual implementation of the adjoint method. However, as we discuss in Section 2.3.9.2, the discrete adjoint method can be directly implemented using reverse AD. In the Julia SciML ecosystem, `ReverseDiffAdjoint` performs reverse AD on the numerical solver via `ReverseDiff.jl`; `ZygoteAdjoint` via `Zygote.jl`; and `TrackerAdjoint` via `Tracker.jl`. In all these cases, a custom pullback function needs to be specified that specifies how VJPs are computed thought the numerical solver (Rackauckas et al. 2021).

2.4.2.2.2 Continuous adjoint method The continuous adjoint method offers a series of advantages over the discrete method and the rest of the forward methods previously discussed. Just as the discrete adjoint methods and backpropagation, the bottleneck is how to solve for the adjoint $\lambda(t)$ due to its dependency with VJPs involving the state $u(t)$.

Effectively, notice that Equation (2.51) involves the terms $f(u,\theta,t)$ and $\frac{\partial h}{\partial u}$, which are both functions of $u(t)$. In opposition to the discrete adjoint methods, notice that here the full continuous trajectory $u(t)$ is needed, instead of its discrete pointwise evaluation. There are two solutions for addressing the evaluation of $u(t)$ during the backwards step.

(i) **Interpolation.** During the forward model, we can store in memory intermediate states of the numerical solution allowing the dense evaluation of the numerical solution at any given time. This can be done using dense output formulas, for example by adding extra stages to the Runge-Kutta scheme (Equation (2.2)) that allows to define a continuous interpolation, a method known as continuous Runge-Kutta (Alexe et al. 2009; Wanner et al. 1996).

(ii) **Backsolve.** Solve again the original system of ODEs together with the adjoint as the solution of the reversed augmented system (Chen et al. 2018)

$$\frac{d}{dt}\begin{bmatrix} u \\ \lambda \\ \frac{dL}{d\theta} \end{bmatrix} = \begin{bmatrix} -f \\ -\frac{\partial f}{\partial u}^T \lambda - \frac{\partial h}{\partial u}^T \\ -\lambda^T \frac{\partial f}{\partial \theta} - \frac{\partial h}{\partial \theta} \end{bmatrix} \qquad \begin{bmatrix} u \\ \lambda \\ \frac{dL}{d\theta} \end{bmatrix}(t_1) = \begin{bmatrix} u(t_1) \\ \frac{\partial L}{\partial u(t_1)} \\ \lambda(t_0)^T s(t_0) \end{bmatrix}. \tag{2.66}$$

An important problem with this approach is that computing the ODE backwards $\frac{du}{dt} = -f(u,\theta,t)$ can be unstable and lead to large numerical errors (Kim et al. 2021; Zhuang et al. 2020). One way of solving this system of equations that ensures stability is by using implicit methods. However, this requires cubic time in the total number of ordinary differential equations, leading to a total complexity of $\mathcal{O}((n+p)^3)$ for the adjoint method.

Both interpolating and backsolve adjoint methods can be implemented along with a checkpointing scheme. This is implemented in `Checkpointing.jl` (Schanen et al. 2023).

When dealing with stiff differential equations, special considerations need to be taken into account. Two alternatives are proposed in (Kim et al. 2021), the first referred to as *Quadrature Adjoint* produces a high order interpolation of the solution $u(t)$ as we move forward, then solve for λ backwards using an implicit solver and finally integrating $\frac{dL}{d\theta}$ in a forward step. This reduces the complexity to $\mathcal{O}(n^3+p)$, where the cubic cost in the number of ODEs comes from the fact that we still need to solve the original stiff differential equation in the forward step. A second similar approach is to use an implicit-explicit (IMEX) solver, where we use the implicit part for the original equation and the explicit for the adjoint. This method also has a complexity of $\mathcal{O}(n^3+p)$.

2.4.2.2.3 Solving the quadrature Another computational consideration is how the integral in Equation (2.53) is numerically evaluated. Some methods save computation by noticing that the last step in the continuous adjoint method for evaluating $\frac{dL}{d\theta}$ is an integral instead of an ODE, and then it can be computed without the need to include it inside the numerical solver (Kidger et al. 2021). Numerical integration, also known as quadrature integration, consists in approximating integrals by finite sums of the form

$$\int_{t_0}^{t_1} F(t) dt \approx \sum_{i=1}^{K} \omega_i \, F(\tau_i), \tag{2.67}$$

where the evaluation of the function occurs in certain knots $t_0 \leq \tau_1 < \ldots < \tau_K \leq t_1$, and ω_i are weights. Weights and knots are obtained in order to maximize the order in which polynomials are exactly integrated (Stoer et al. 2002). Different quadrature methods are based on different choices of the knots and associated weights. Between these methods, the Gaussian quadrature is the faster method to evaluate one-dimensional integrals (Norcliffe et al. 2023), which gives the name to the Gaussian adjoint method in Table 2.1.

2.5 Conclusions

In the present work, we presented a comprehensive overview of the different existing methods for calculating the sensitivity or gradients of forward maps involving numerical solutions of differential equations. This task has been approached from three different angles. First, we presented the existing literature in different scientific communities where adjoints and sensitivities have been used before and play a central modelling role, especially for inverse modeling. Secondly, we reviewed the mathematical foundations of these methods and their classification as forward-reverse and discrete-continuous. Finally, we have shown how the different methods are implemented in Julia and the different computational considerations that we must take into account when implementing or using a sensitivity algorithm.

Software availability. The GitHub repository `DiffEqSensitivity-Review` contains both text and code used to generate this manuscript. The manuscript for the review paper and the bibliography file are automatically generated with GitHub actions. See Appendix A for a complete description of the scrips provided. The symbol ♣ will be use to reference code associated with scripts in the repository.

Chapter 3

Glacier modelling

In this section we are going to introduce some basic elements in glacier ice flow modelling. Ice on Earth can be mostly found in the two polar ice caps, Antarctica and Greenland, and distributed around the approximately $274,000$ mountain glaciers around the globe (RGI 7.0 Consortium 2023). Ice shelves, ice sheets, and mountain glaciers can be modelled as a very slow viscous fluid, pushed by gravity and internal hydro-static pressure, in interaction with the surrounding rock via friction mechanisms, on top of which accumulation and ablation effects (including melting and calving) occur.

The two most common approaches found in the literature to model glacier ice flow are the following:

1. **Mass conservation approach.** Simple, but accurate. This consists on assuming that the glacier is in a balance state, that is, the flow of ice balances the ablation (loss of ice) and accumulation of new ice. This approach fails when the glacier is far from a steady state, but it also predicts many observed characteristics of glacier flow.

2. **Mechanical physics.** The gold standard, this is what we want to use in theory. Here, we try to understand the evolution of a glacier based on the effect of the gravitational force plus the boundary conditions of the glacier. Models based on mechanical physics approaches are more difficult to implement and lack a good data assimilation pipeline, potentially leading to useless predictions. However, they have the potential to better describe the physical processes involved in ice flow.

In this chapter we are going to derive the fundamental differential equations governing ice flow. These are going to be the governing laws we will use to describe the forward model of the workflow in ODINN in Chapter 4.

3.1 Physical foundations

Let us first begin with some notation. The standard convention in glacier modelling is that coordinate x is used to describe the flowline direction; y the cross-sectional direction; and

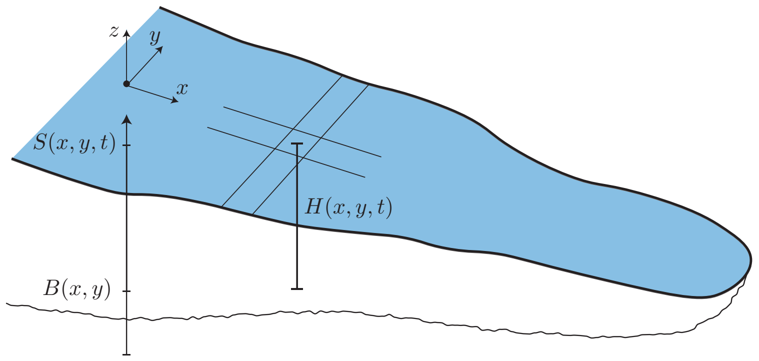


Figure 3.1: *Diagram of a mountain glacier with its associated coordinate system.*

z is used to described the vertical axes pointing in the direction of gravity (see Figure 3.1). This applies very well for elongated or tongue-shaped glaciers, where the scale of the physical processes applying in the direction of flow are different than the ones occurring in the cross-sectional direction of ice flow. In general, both x and y describe a generic planar coordinate system. We further introduce the following shape variables:

- $B(x, y)$ the bed topography, that is, the altitude of the rock underlying the glacier.
- $H(x, y, t)$ the height of the ice column of ice.
- $S(x, y, t) = B(x, y) + H(x, y, t)$ the ice surface elevation.

The only dynamical quantities we consider here are the shape of the glacier, which affects the ice thickness H and the observed surface elevation S. When redundant, we will ignore the temporal and spatial dependency on these variables.

3.1.1 Continuity equation

We will focus on the modelling of changes in the height of the ice column $H(x, y, t)$ instead of the full three-dimensional flow of ice. This strategy involves deriving depth-integrated equations, meaning that we sum up all the sources of stresses/forces acting on a column of ice to come up with a vertically averaged description of the glacier. The variation in ice thickness is determined by the continuity equation (Cuffey et al. 2010). If $\rho(x, y, z, t)$ and $u(x, y, z, t)$ denote the density of ice and ice flow velocity, respectively, then the continuity equation for fluids states (Landau et al. 2013)

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0. \tag{3.1}$$

For a constant density and without melting or aggregation of ice, the continuity equation reduces pointwise to the incompressibility condition $\nabla \cdot u = 0$. Assuming a constant density

ρ is justified by the fact that the superficial layer known as firn where the density varies is relatively small compared to the rest of the glacier (Fowler 2010). However, since addition and removal of ice happen just in the bottom and surface of a glacier, we instead consider the vertically integrated version of the continuity equation. The vertical integration of the continuity equation results in a new continuity equation of the mass change over the vertical column of ice with length $H(x, y, t)$. Now, we can integrate this last equation in order to have the mass change for a vertical column of ice. Considering the boundary conditions related to the vertical velocities in both the bed and the surface (Whillans 1977), we obtain the following version of the continuity equation for constant density ρ,

$$\frac{\partial H}{\partial t} = \dot{b} - \nabla_{xy} \cdot q, \tag{3.2}$$

with q the ice flow given by

$$q(x, y, t) = \int_{B(x,y)}^{S(x,y,t)} u(x, y, z) dz = \bar{u} H(x, y, t), \tag{3.3}$$

and ∇_{xy} the divergence on the horizontal plane (when referring to two dimensional flows, we will use the symbol ∇ to refer to ∇_{xy}). The term $\dot{b} = \dot{b}(x, y, t)$ is known as mass balance and represents addition or removal of ice happening both in the surface and bottom of the glacier. Since most of the removal and aggregation of ice happens in the surface instead of in the bed, we usually will model only the surface mass balance (SMB) contribution and we will call $\dot{b}$ both surface mass balance or just mass balance (MB).

The continuity equation states that solving for the ice thickness H and solving for the ice flow u are the same problem once we know the mass balance. It is also interesting to notice that variations in ice thickness are ruled by two different processes: the mass balance term associated to accumulation and ablation of ice (usually related to climatic or thermodynamic mechanisms) and the ice dynamical term that establishes how the mass of ice is re-distributed as a consequence of the movement of ice.

The *mass conservation approach* described previously consists in assuming that changes in ice thickness H are very slow compared to the processes of flow and mass balance in Equation (3.2). This is certainly the case of a glacier in equilibrium, where the shape of the glacier does not change drastically over time. For these cases, the continuity equation reads as

$$\dot{b} \approx \nabla_{xy} \cdot q. \tag{3.4}$$

This last equation tell us that we can inform changes in ice velocities (dictated by physical mechanisms) based on observations of accumulation (dictated by climate and weather).

If useful and simple to perform, the mass conservation approach relies on the strong hypothesis of stationary glaciers that do not drastically change over time. Considering the rapid retreat of mountain glaciers all around the world, more complex and complete approaches are needed in order to capture the fast variability of glacier shape. In the next section, and in the rest of this thesis, we will instead consider the mechanical physics approach.

3.1.2 Glen's law

Glen's Law is a phenomenological law that relates shearing $\dot{\epsilon}$ with deviatoric stress τ using a power law $\dot{\epsilon} = A\tau^n$ (Glen 1955). The coefficients of this power law relation are the Glen exponent n and Glen coefficient A, also known as creep exponent and creep parameter. Importantly for our analysis, these coefficients are generally non-constant and may vary both temporally and spatially. Although it is usually assumed and there is some evidence supporting that $n \approx 3$, this number can vary between $n \approx 1.8$ and $n \approx 4$ for different configurations of ice, including the stress configurations and the ice fabric, that is, the grain size and distribution of ice crystals, as the main driver (Behn et al. 2021). Furthermore, the viscosity of the ice and consequently the Glen parameter A are affected by multiple factors, including ice temperature, pressure, water content, and ice fabric (Cuffey et al. 2010). For example, ice is harder and therefore more resistant to deformation at lower temperatures. The rest of this section is dedicated to the mathematical formulation of the Glen creep law using tensorial analysis.

When mathematically modeling a fluid, we associate different tensors to each differential unit, in our case, a small differential cube of ice. A tensor is a algebraic object that is invariant under change of the coordinate system. The Jacobian of the velocity vector $u = (u_1, u_2, u_3)$ (first order tensor) is the second rank tensor given by

$$\begin{pmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_2}{\partial x} & \frac{\partial u_3}{\partial x} \\ \frac{\partial u_1}{\partial y} & \frac{\partial u_2}{\partial y} & \frac{\partial u_3}{\partial y} \\ \frac{\partial u_1}{\partial z} & \frac{\partial u_2}{\partial z} & \frac{\partial u_3}{\partial z} \end{pmatrix}. \tag{3.5}$$

All tensors can be decomposed as the sum of a symmetric and antisymmetric (or skew-symmetric) tensor (Spain 2003; Synge et al. 1978). In physical terms, the symmetric component describes the stretching and shearing while the second component quantifies the rate of rotation. Since antisymmetric matrices can be used to compute cross products as matrix representations, it is easy to check that the antisymmetric part corresponds to a solid rotation around the vorticity vector. On the other hand, the rate of deformation is encoded in the symmetric part of the speed tensor, given by

$$\dot{\epsilon}_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \tag{3.6}$$

Here we had introduced the common notation $x_1 = x$, $x_2 = y$ and $x_3 = z$. Now, the diagonal terms of the tensor $\dot{\epsilon}$ correspond to the stretching of an unit of ice, while the non-diagonal terms quantify shearing. For an incompressible fluid we have that the total stretching is zero, meaning $\sum_i \dot{\epsilon}_{ii} = 0$.

The second important tensor we need to consider to model ice deformation is the stress tensor. The stress tensor σ is a second rank tensor that allow us to compute the stress/pressure that a differential of ice experiences. Given a normal vector $n = (n_x, n_y, n_z)$, the stress T in the direction n is a vector that represents the force by unit of area that a differential of ice is experiencing in that direction and can be computed as

$$\begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}. \tag{3.7}$$

If the non diagonal terms in σ are non-zero, this means that a piece of ice will shrink in one direction. Another important property is that σ is a symmetric tensor, meaning that $\sigma_{xy} = \sigma_{yx}$, $\sigma_{xz} = \sigma_{zx}$ and $\sigma_{yz} = \sigma_{zy}$. This is a necessary condition in order to satisfy the torque balance of an infinitesimal cube of ice (Fowler 2010).

It is the deviatoric stress τ that determines the rate of deformation of ice. The *deviatoric stress* tensor τ is defined as

$$\tau = \begin{pmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{pmatrix} = \begin{pmatrix} \sigma_{xx} - \sigma_M & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} - \sigma_M & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} - \sigma_M \end{pmatrix}, \tag{3.8}$$

with σ_M the normal mean stress defined as $\sigma_M = \frac{1}{3}(\sigma_{xx}+\sigma_{yy}+\sigma_{zz})$, which also coincides with the negative of the mean pressure $p = -\sigma_M$ (Cuffey et al. 2010; Fowler et al. 2020). Glen's law relates the rate of deformation of ice with deviatoric stress via the power relationship $\dot{\epsilon} = A(\tau)\tau^n$. The isotropic condition of ice imposes the constraint $\epsilon_{ij} = \lambda(\tau)\tau_{ij}$, with $\lambda(\tau)$ a function of an invariant of the tensor τ. This leads to the following tensorial form of the Nye-Glen's law,

$$\dot{\epsilon}_{jk} = A\,\tau_E^{n-1}\,\tau_{jk}. \tag{3.9}$$

with

$$\tau_E^2 = \frac{1}{2}\left(\tau_{xx}^2 + \tau_{yy}^2 + \tau_{zz}^2\right) + \tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2. \tag{3.10}$$

the effective stress; and A the Glen coefficient.

Solving for the deviatoric tensor instead we obtain $\tau_{jk} = \eta\dot{\epsilon}_{jk}$, with η the viscosity of the fluid given by $\eta = 1/A\tau_E^{n-1}$. For $n = 3$, the viscosity decreases if we increase the stress at any direction. Ice becomes more fluid when the stress increases. The creep of glacier ice is intermediate between Newtonian viscous and perfect plastic behaviours.

3.1.3 Flow equations

The *mechanical physics approach* consists of deriving differential equations for the variations in glacier shape and flow, represented by the ice thickness profile or the ice flow velocity field. The Navier-Stokes equation for viscous fluids is

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla p + \nabla \cdot (\eta \nabla u) + \rho g. \tag{3.11}$$

with $p = p(x, y, z, t)$ the pressure; $\eta = \eta(x, y, z, t)$ the viscosity; and $g = g(z)$ the gravitational acceleration (Brenner 2005; Lemarié-Rieusset 2018). Now, ice behaves as a slow fluid that does not experience turbulence, convection, or the Coriolis force like the atmosphere and ocean (Bueler 2014). This means that the left side of Equation (3.11) is approximately zero in comparison with the right side. More technically, this simplification is justified by the fact that ice behaves as a viscous fluid with Reynolds number of the order of 10^{-14} (Fowler et al. 2020). The remaining equation is just the balance of forces, and combined with the incompressibility of ice and Glen's law, they define the set of Stokes equations (Bueler 2014):

$$\nabla \cdot u = 0 \tag{3.12}$$
$$0 = -\nabla p + \nabla \cdot \tau_{ij} + \rho g \tag{3.13}$$
$$\dot{\epsilon}_{ij} = A \tau_E^{n-1} \tau_{ij}. \tag{3.14}$$

Notice that Stokes equations do not contain any time derivative. This means that the boundary conditions, the gravity force, and other parameters alone determine the velocity and stress fields. Velocity is a diagnosis or output of ice flow models, without memory of prior velocity. However, this does not mean that the velocity field of a glacier is constant over time. Ice flux can change the shape of the glacier (for example, its thickness) which may impact in the velocity field.

Now, these set of equations are too difficult to handle in general, especially since they are still modelling the full three-dimensional flow. Two common simplified depth-integrated models are given by:

- Shallow Ice Approximation (SIA)
- Shallow Shelf Approximation (SSA)

As the names indicate, they are both *shallow* approximations, meaning that we assume thin glaciers, with small depth to width ratio. This applies in general to ice sheets and ice streams. They are both depth-integrated methods, meaning that we end up with a set of equations that depend on x and y but not on z.

However, the SIA and SSA differ in the approximations they make about the components of the deviatoric stress tensor

$$\tau = \begin{pmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ & \tau_{yy} & \tau_{yz} \\ & & \tau_{zz} \end{pmatrix}. \tag{3.15}$$

In SIA, we neglect the terms τ_{xx}, τ_{yy} and τ_{xy} (and then τ_{zz}) and we just consider τ_{xz} and τ_{yz} as non-zero components. This implies that the driving stress caused by gravity is only balanced by the basal resistance τ_b. The SIA assumption has been first developed and it holds very well for mountain glaciers, with the caveat that it fails in regions close to the grounding line and it is not able to reproduce short time scale events (Bueler et al. 2009). On the contrary, the SSA assumes that τ_{xz} and τ_{yz} are negligible with respect to the rest of the components of τ_{ij}. This implies that SSA considers both longitudinal resistive forces and lateral shearing, which is a valid assumption for ice shelves and low-drag ice streams.

Notice that in SIA, since the diagonal of τ_{ij} is zero, then $\frac{\partial u_x}{\partial x}$, $\frac{\partial u_y}{\partial y}$ and $\frac{\partial u_z}{\partial z}$ are necessary zero. In SSA, in principle all components of the velocity vector u can depend on x, y and z.

In the next section, we are going to derive the SIA equation. For the interested reader, the mathematical derivation of the SSA can be found in Appendix B.

3.1.4 Shallow ice approximation (SIA)

We are going to start with out analysis of the Shallow Ice Approximation (SIA). Let us derive an expression for the vertical profile of the velocity of a glacier and an expression for the surface velocity. We are going to assume that the glacier deforms in simple shear, a situation called *laminar flow*. Without loss of generality, we can assume $\tau_{yz} = 0$. If $\tau_{yz} \neq 0$ the final result is exactly the same, but we need to consider the direction in which the ice column is moving in the (x, y) plane. The deviatoric stress is given by

$$\tau = \begin{pmatrix} 0 & 0 & \tau_{xz} \\ 0 & 0 & 0 \\ \tau_{xz} & 0 & 0 \end{pmatrix}, \tag{3.16}$$

where the component τ_{xz} is a linear interpolation consequence of hydrostatic pressure between $\tau_{xz} = 0$ in the surface of the glacier and $\tau_{xz} = \tau_b$ at the bed of the glacier, with τ_b the basal drag, that is,

$$\tau_{xz} = \tau_b \left(1 - \frac{z}{H}\right). \tag{3.17}$$

Based on Glen's law we have

$$\frac{du}{dz} = 2A\tau_{xz}^n = 2A\tau_b^n \left(1 - \frac{z}{H}\right)^n, \tag{3.18}$$

where $z = 0$ coincides with the bed. Integrating last expression from the bed to a generic value of z gives the vertical profile

$$\begin{aligned} u(z) - u_b &= 2A\tau_b^n \int_0^z \left(1 - \frac{z}{H}\right)^n dz \\ &= \frac{2A}{n+1}\tau_b^n H \left[1 - \left(1 - \frac{z}{H}\right)^{n+1}\right], \end{aligned} \tag{3.19}$$

with u_b the velocity at the bed. Finally, the flux per column of ice results from integrating $u(z)$ from the bed to the surface results in (Monnier et al. 2017)

$$\begin{aligned} q = H\bar{u} &= \int_0^H u(z)dz = Hu_b + \frac{2A}{n+2}H^2\tau_b^n \\ &= \left(C + \frac{2A}{n+2}H\right) H^2\tau_b^n. \end{aligned} \tag{3.20}$$

where we defined the constant C to encapsulates the contribution of the basal flow u_b to the flux.

Using the equilibrium of hydrostatic forces that a column of ice experiences, we can derive the commonly used expression for the basal drag $\tau_b = -\rho g H \alpha$, with $\alpha = \|\nabla S\|$ the slope on the surface. Then we can write

$$\bar{u}(x,t) = -\left(C + \frac{2A}{n+2}H\right)(\rho g)^n H^{n+1}\|\nabla S\|^{n-1}\nabla S. \tag{3.21}$$

The final step to derive the SIA equations is to replace the expression for the flow $q = H\bar{u}$ in Equation (3.21) in the continuity equation (3.2),

$$\frac{\partial H}{\partial t} = \dot{b} + \nabla \cdot \left(\left(C + \frac{2A}{n+2}H\right)(\rho g)^n H^{n+1}\|\nabla S\|^{n-1}\nabla S\right), \tag{3.22}$$

where n and A are the creep exponent and parameter in Glen's Law, respectively; $\dot{b}$ is the mass balance (MB); C is a basal sliding coefficient; and ∇S is the gradient of the glacier surface $S(x,y,t) = B(x,y) + H(x,y,t)$, with $B(x,y)$ the bedrock elevation, and $\|\nabla S\|$ denotes its Euclidean norm (Monnier et al. 2017). The operator $\nabla = \nabla_{xy}$ here includes just the spatial derivatives in x and y. Now, we can rewrite this last equation in a more compact form as

$$\frac{\partial H}{\partial t} = \dot{b} + \nabla \cdot (D\nabla S), \tag{3.23}$$

where D is the *effective diffusivity* given by

$$D = D(H, \nabla S; A, n, C) = \left(C + \frac{2A}{n+2}H\right)(\rho g)^n H^{n+1}\|\nabla S\|^{n-1}. \tag{3.24}$$

This last differential equations are analogous to the heat equation for $H(x,y,t)$, where the diffusivity coefficient depends on H.

A convenient simplification of the SIA equation is to assume $C = 0$, which implies the basal velocity is zero all along the bed. This is reasonable when large portions of the glacier bed experience minimal sliding. In that case, the effective diffusivity D is given by

$$D = \Gamma\, H^{n+2}\|\nabla S\|^{n-1}, \quad \Gamma = \frac{2A}{n+2}(\rho g)^n. \tag{3.25}$$

An important property of the SIA equation is that the ice surface velocity u can be directly derived from the ice thickness H by the equation

$$u = -\frac{2A}{n+1}(\rho g)^n H^{n+1}\|\nabla S\|^{n-1}\nabla S. \tag{3.26}$$

This relationship is particularly useful for physical inversions, where ice thickness can be inverted from ice surface velocities observations (Millan et al. 2022).

3.2 Numerical solutions

Except for a few engineered initial glacier conditions, no analytical solutions for the SIA differential equation (Equation (3.23)) are known, and it is necessary to use numerical methods in order to find approximate solutions (Halfar 1981). A short introduction of numerical solvers for ordinary differential equations was covered in Section 2.3.1.1.

Numerically solving the SIA equation is challenging for the following reasons.

1. The diffusivity D depends on the unknown solution H. In most cases, the diffusivity involved in the heat equation is a spatial-temporal function, but not a function of the solution itself.
2. The SIA equation is a stiff differential equation (Wanner et al. 1996). This is because the diffusivity D is very sensitive to changes in the shape of the glacier. For example, for $n = 3$, we have $D \propto H^5 \|\nabla S\|^2$. Values of the diffusivity D for different values of H and slopes $\alpha = \|\nabla S\|$ are shown in Figure 3.2.
3. The SIA equation is a *degenerate diffusion equation* since the diffusivity D vanishes as $H \to 0$ or $\nabla S \to 0$ (Bueler 2014).
4. Non-negativity of the solution. Although the continuous differential equation (3.23) satisfies $H \geq 0$ for all times, this constraint may be violated by numerical solvers based on discrete time updates. This means that the numerical solver needs to further impose the constraint $H \geq 0$.

Both items 1 and 2 can be handled with a staged grid (Section 3.2.1) and a numerical solver with stepsize controllers that guarantee low numerical error at the cost of small stepsizes (Section 2.3.1.1). Item 3 is easily handled by treating the SIA equation as a free-boundary problem, which is allowed as long as the glacier does not extend to the margins of our domain (Fowler et al. 2020). Item 4 is enforced by adding an extra stepsize controller to the numerical solver and clipping gradients when necessary (Section 3.2.2) (Imhof 2021). All these methods are implemented inside the numerical solvers in `ODINN.jl`, which we will cover in more detail in Chapter 4.

Just to gain some intuition of ice flow before dealing with some more technical aspects of the numerical solver, Figure 3.3 shows the results of solving the SIA equation for the case of a dome inspired glacier lying on a flat bed. As we move forward in time (in this case a total amount of time of 30 years) we see how the ice spreads on the sides and smooths out the extreme curvature of the initial condition.

3.2.1 Gridding

The strategy to solve the SIA partial differential equation follows the *method of lines*, where the spatial coordinate is discretized and the PDE is transformed into a system of sparse coupled ordinary differential equations (ODEs) (Ascher 2008). We are going to consider a

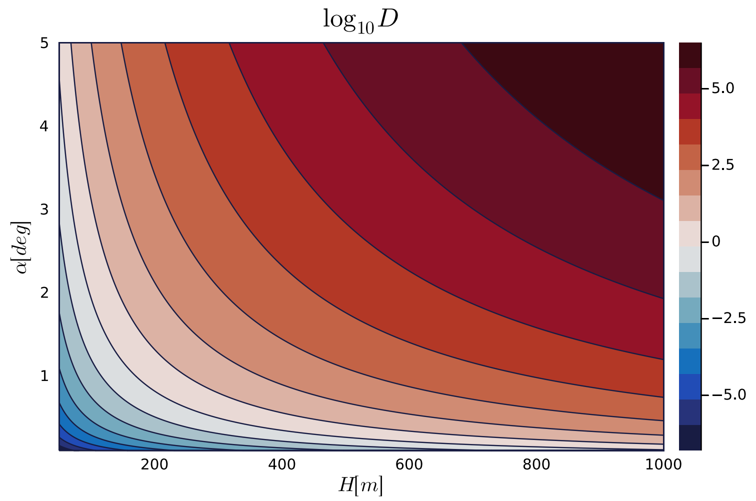


Figure 3.2: *Different values of the effective diffusivity D (units of m^2/yr) as a function of the ice thickness H and surface slope $\alpha = \|\nabla S\|$.*

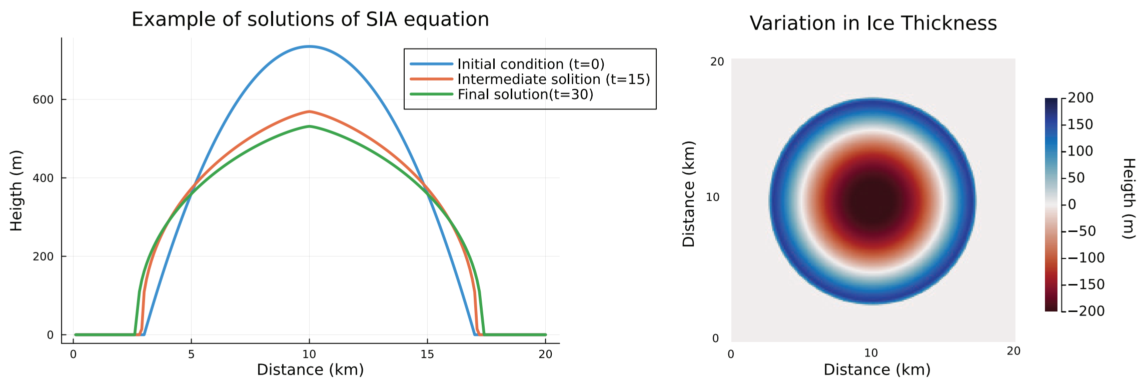


Figure 3.3: *Simulation of the SIA equation for a synthetic example with a flat bed.*

uniform grid on points (x_j, y_k), with $j = 1, 2, \dots, N_x$ and $k = 1, 2, \dots, N_y$, with $\Delta x = \Delta y = x_{j+1} - x_j = y_{k+1} - y_k$. Then each $H_{j,k} = H(t, x_j, y_k)$ is just a function of time. Starting from some initial time t_0, we are going to update the value of the solution for H by steps Δt_i, with $t_i = t_{i-1} + \Delta t_{i-1}$. We are going to refer as $H^i_{j,k}$ for the numerical approximation of $H(t_i, x_j, y_k)$. In this way, we are going to have a system of ODEs for all the $H_{j,k}$.

An important consideration when working with numerical schemes for differential equations is the stability of the method (Hairer et al. 2008). Here we are going to consider just explicit methods, although the spatial discretization is the same for implicit methods. Explicit methods for the SIA equation are conditionally stable, meaning that stability is guaranteed given the following conditions (Fowler et al. 2020).

1. Evaluation of the diffusivity in a staggered grid $D_{i\pm\frac{1}{2},k\pm\frac{1}{2}}$ labeled by semi-integer indices (circles on the dotted grid in Figure 3.4). This grid coincides with Scheme E of Arakawa grids (Arakawa et al. 1977).

2. Usage of a timestep controller such that $\Delta t_i \leq \Delta x^2/4D^i_{\max}$, where $D_{\max}$ is the maximum diffusivity at step i.

The algorithm to solve the SIA equation follows the next one-step iterative procedure:

1. Given the value of $H_{j,k} = H^i_{j,k}$ at some give time t_i, compute the value of the diffusivity on the staggered grid. As we mentioned before, the diffusivity is a function of H, $\frac{\partial S}{\partial x}$ and $\frac{\partial S}{\partial y}$. Instead of using one single estimate of H to approximate all these quantities, the idea is to use averaged quantities on the primal grid to compute the diffusivity on the staggered grid (red arrows in Figure 3.4). We define the average quantities

$$H_{j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{4}\left(H_{j,k} + H_{j+1,k} + H_{j,k+1} + H_{j+1,k+1}\right) \tag{3.27}$$

$$\left(\frac{\partial S}{\partial x}\right)_{j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{2}\left(\frac{H_{j+1,k} - Hj,k}{\Delta x} + \frac{H_{j+1,k+1} - H_{j,k+1}}{\Delta x}\right) \tag{3.28}$$

$$\left(\frac{\partial S}{\partial y}\right)_{j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{2}\left(\frac{H_{i,k+1} - H_{i,k}}{\Delta y} + \frac{H_{i+1,k+1} - H_{i+1,k}}{\Delta y}\right). \tag{3.29}$$

Then, we compute the diffusivity on the staggered grid as

$$D_{j+\frac{1}{2},k+\frac{1}{2}} = D\left(H_{j+\frac{1}{2},k+\frac{1}{2}}, \left(\frac{\partial S}{\partial x}\right)_{j+\frac{1}{2},k+\frac{1}{2}}, \left(\frac{\partial S}{\partial y}\right)_{j+\frac{1}{2},k+\frac{1}{2}}\right). \tag{3.30}$$

2. Compute a different average diffusivity but now on the edges of the primal grid (blue arrows in Figure 3.4):

$$D_{j,k\pm\frac{1}{2}} = \frac{1}{2}\left(D_{j-\frac{1}{2},k\pm\frac{1}{2}} + D_{j+\frac{1}{2},k\pm\frac{1}{2}}\right), \tag{3.31}$$

$$D_{j\pm\frac{1}{2},k} = \frac{1}{2}\left(D_{j\pm\frac{1}{2},k-\frac{1}{2}} + D_{j\pm\frac{1}{2},k+\frac{1}{2}}\right) \tag{3.32}$$

3. Compute the diffusive part of the SIA equations on the point in the primal grid (j,k) as

$$\begin{aligned}\nabla(D\nabla S)_{j,k} = &\frac{D_{j+\frac{1}{2},k}(S_{j+1,k} - S_{j,k}) - D_{j-\frac{1}{2},k}(S_{j,k} - S_{j-1,k})}{\Delta x^2} \\ &+ \frac{D_{j,k+\frac{1}{2}}(S_{j,k+1} - S_{j,k}) - D_{j,k-\frac{1}{2}}(S_{j,k} - S_{j,k-1})}{\Delta y^2}.\end{aligned} \tag{3.33}$$

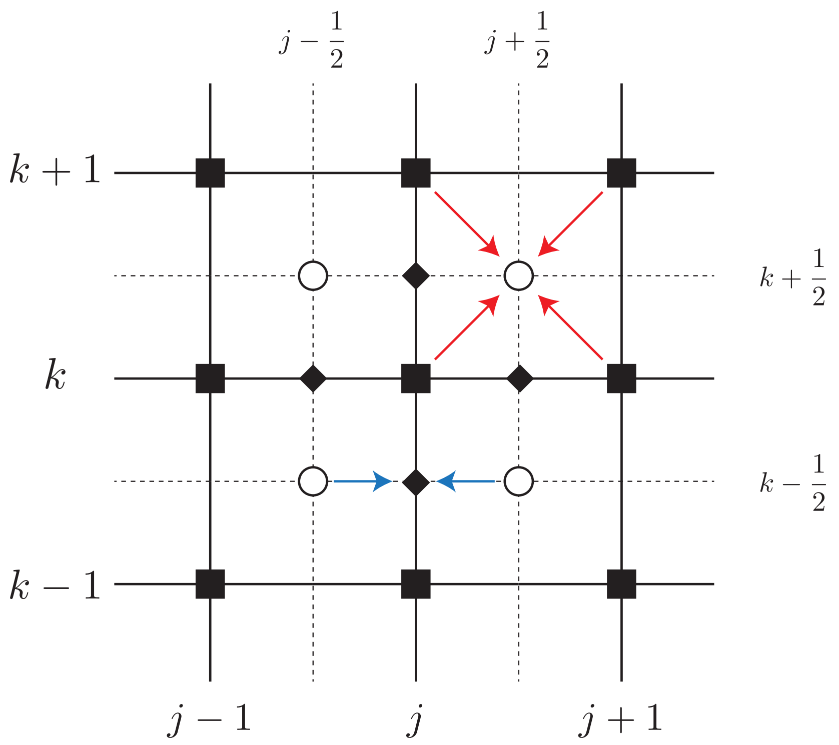


Figure 3.4: *Staggered grid used to solve the Shallow Ice Approximation PDE. Black squares represent the primal grid; empty circles the staggered grid; diamonds represent the points in the grid where the diffusivity evaluated in the staggered grid is averaged (Equations (3.31) and (3.32)); blue arrows operations in the edges of the primal grid, and red arrows operations in the staggered grid.*

4. Update the value of H following an explicit or implicit scheme (see Section 2.3.1.1). Just for illustration, the explicit Euler method will update the values of the ice thickness following

$$H_{j,k}^{i+1} = H_{j,k}^{i} + \Delta t_i \left(\dot{b}_{j,k}^{i} + \nabla (D \nabla S)_{j,k}^{i} \right) . \tag{3.34}$$

where Δt_i is the time step, which in our case is automatically selected by the numerical solver to ensure stability, for example $\Delta t_i = \eta \Delta x^2 / 4 D_{\text{max}}^i$ with $\eta \in (0,1)$.

3.2.2 Algebraic constraint

A priori, the stepsize Δt_i is chosen by a stepsize controller integrated inside the numerical solver. The purpose of these is to pick the stepsize as large as possible (so the solver requires less total steps) at the same time that local errors induced by the numerical solver are bounded (Hairer et al. 2008; Ranocha et al. 2022). However, this does not automatically guarantee that the updates in the ice thickness (Equation (3.34)) make $H_{j,k}^{i+1} \geq 0$. A sufficient condition for $H_{j,k}^{i+1} \geq 0$ is given by (Imhof 2021)

$$-H_{j,k}^{i} \leq S_{j\pm1,k} - S_{j,k}, \tag{3.35}$$

$$-H_{j,k}^{i} \leq S_{j,k\pm1} - S_{j,k}. \tag{3.36}$$

This condition guarantees that the computed diffusivity (Equation (3.33)) satisfies

$$\nabla(D\nabla S)_{j,k}^{i} \geq -\frac{4D_{\max}}{\Delta x^2} H_{j,k}^{i} \tag{3.37}$$

and hence

$$H_{j,k}^{i+1} \geq H_{j,k}^{i} + \Delta t_i \dot{b}_{j,k}^{i} - \frac{4\Delta t_i D_{\max}}{\Delta x^2} H_{j,k}^{i} = \left(1 - \frac{4\Delta t_i D_{\max}}{\Delta x^2}\right) H_{j,k}^{i} + \Delta t_i \dot{b}_{j,k}^{i} \geq \Delta t_i \dot{b}_{j,k}^{i}, \tag{3.38}$$

where the last inequality is consequence of the stability condition $\Delta t_i \leq \Delta x^2 / 4D_{\max}$. In cases where no mass balance term is added ($b_{j,k}^{i} = 0$), we simply have that $H_{j,k}^{i+1} \geq 0$ for all grid points. In the general case with mass balance, we still need to clip the updated ice thickness by replacing $H_{j,k}^{i+1}$ by $\max\{0, H_{j,k}^{i+1}\}$. This includes those cases with excessive ablation.

Chapter 4

Universal differential equations for glacier ice flow modelling

The contents of this chapter are based on the following:

- J. Bolibar, F. Sapienza, F. Maussion, R. Lguensat, B. Wouters, and F. Pérez (2023a). "Universal differential equations for glacier ice flow modelling". In: *Geoscientific Model Development* 16.22, pp. 6671–6687. DOI: 10.5194/gmd-16-6671-2023[1]
- J. Bolibar and F. Sapienza (June 2023b). *ODINN-SciML/ODINN.jl: v0.2.0*. Version v0.2.0. DOI: 10.5281/zenodo.8033313

The associated publication has highlighted in the journal *Geoscientific Model Development*.

4.1 Abstract

Geoscientific models are facing increasing challenges to exploit growing datasets coming from remote sensing. Universal differential equations (UDEs), aided by differentiable programming, provide a new scientific modelling paradigm enabling both complex functional inversions to potentially discover new physical laws and data assimilation from heterogeneous and sparse observations. We demonstrate an application of UDEs as a proof of concept to learn the creep component of ice flow, i.e. a nonlinear diffusivity differential equation, of a glacier evolution model. By combining a mechanistic model based on a 2D Shallow Ice Approximation partial differential equation with an embedded neural network, i.e. a UDE, we can learn parts of an equation as nonlinear functions that then can be translated into mathematical expressions. We implemented this modelling framework as `ODINN.jl`, a package in the Julia programming language, providing high performance, source-to-source automatic differentiation (AD) and seamless integration with tools and global datasets from the Open Global Glacier Model (OGGM) in Python. We demonstrate this concept for 17

[1]J. Bolibar and F. Sapienza contributed equally to this work.

different glaciers around the world, for which we successfully recover a prescribed artificial law describing ice creep variability by solving ∼500,000 ordinary differential equations in parallel. Furthermore, we investigate which are the best tools in the scientific machine learning ecosystem in Julia to differentiate and optimize large nonlinear diffusivity UDEs. This study represents a proof of concept for a new modelling framework aiming at discovering empirical laws for large-scale glacier processes, such as the variability of ice creep and basal sliding for ice flow, and new hybrid surface mass balance models.

4.2 Introduction

Universal differential equations (UDEs) (Rackauckas et al. 2020)), also known as neural differential equations when using neural networks (Chen et al. 2018; Kidger 2021; Lguensat et al. 2019), combine the physical simulation of a differential equation using a numerical solver with machine learning (Figure 1.2). Optimization algorithms based on exact calculations of the gradient of the loss function require a fully differentiable framework, which has been a technical barrier for some time. Python libraries such as PyTorch (Paszke et al. 2019), Tensorflow (Abadi et al. 2016) or JAX (Bradbury et al. 2020) require rewriting the scientific model and the solver with the specific differentiable operations of each library, making it very costly to apply it to existing models or making solutions very library-centered. Alternatively, the Julia programming language (Bezanson et al. 2017), designed specifically for modern scientific computing, has approached this problem in a different manner. Instead of using library-specific differentiable operators, it performs automatic differentiation (AD) directly on source code. This feature, together with a rich differential equations library provides a suitable scientific machine learning ecosystem to explore new ways to model and understand physical systems (Rackauckas et al. 2019).

In terms of data assimilation and model parameter calibration, many different approaches to obtain differentiable glacier models have been developed (Brinkerhoff et al. 2016; Goldberg et al. 2013; MacAyeal 1993). These inverse modelling frameworks enable the minimization of a loss function by finding the optimal values of parameters via gradient descent. Such gradients can be found by either computing the associated adjoint or by using AD (see Chapter 2). Nonetheless, all efforts so far have been applied to the inversion of scalar parameters and sometimes their distributions (in the context of Bayesian inference), i.e. parameters that are stationary for a single inversion given a dataset. This means that the potential of learning the underlying physical processes is reduced to the current structure of the mechanistic model. No changes are made to the equations themselves, with the main role of the inversions being the fitting of one or more parameters already present in the equations. To advance beyond scalar parameter inversions, more complex inversions are required, shifting towards functional inversions. Functional inversions enable the capture of relationships between a parameter of interest and other proxy variables, resulting in a function that can serve as a law or parametrization. These learnt functions can then be added in the currently existing equation, thus expanding the underlying model with new

ØDINN
ØGGM + DIfferential equation Neural Networks
Open Global Glacier Model

Figure 4.1: *Logo of `ODINN.jl`. The ecosystem of ODINN has been casually and without explicit intention being built around characters in Norse mythology. This includes Odin but also the sub-packages Huginn, Muninn, and Sleipnir.*

knowledge.

We present an application of universal differential equations, i.e. a differential equation with an embedded function approximator (e.g. a neural network). For the purpose of this study, the neural network is used to infer a prescribed artificial law determining the ice creep coefficient in Glen's law based on a climate proxy. Instead of treating it as classical inverse problem, where a global parameter is optimized for a given point in space and time, neural networks learn a nonlinear function that captures the spatiotemporal variability of that parameter. This opens the door to a new way of learning parametrizations and empirical laws of physical processes from data. This case study is based on `ODINN.jl` v0.2 (Bolibar et al. 2023b), a new open-source Julia package, available on GitHub at `https://github.com/ODINN-SciML/ODINN.jl`. With this study, we attempt to share and discuss what are the main advances and difficulties in applying UDEs to more complex physical problems, we assess the current state of differentiable physics in Julia, and we suggest and project the next steps in order to scale this modelling framework to work with large scale remote sensing datasets.

4.3 Methods

The combination of Python tools from OGGM with the UDE glacier modelling framework in Julia has resulted in the creation of a new Julia package named `ODINN.jl` (OGGM + DIferential equation Neural Networks; (Bolibar et al. 2023b)). For the purpose of this study, ODINN has been used to study the viability of UDEs to solve and learn subparts of the SIA equation.

The general overview of `ODINN.jl`'s workflow to perform functional inversions of glacier physical processes is shown in Figure 4.2. The parameters θ of a function determining a given physical process D_θ, expressed by a neural network NN_θ, are optimized in order to minimize a loss function. For this study, the physical law was constrained only by climate data, but

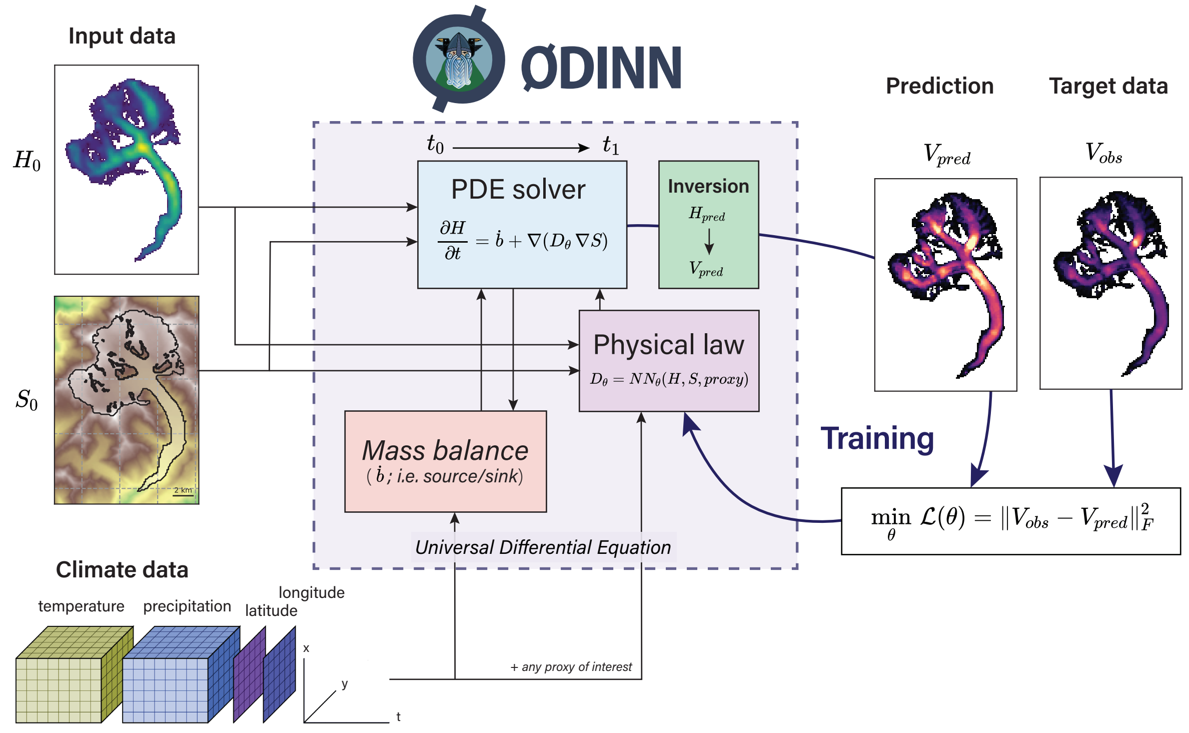


Figure 4.2: *Overview of ODINN.jl's workflow to perform functional inversions of glacier physical processes using Universal Differential Equations. The parameters (θ) of a function determining a given physical process (D_θ), expressed by a neural network NN_θ, are optimized in order to minimize a loss function. For this study, the physical law was constrained only by climate data, but any other proxies of interest can be used to design it. The glacier surface mass balance is downscaled (i.e. it depends) on S, which is updated by the solver, thus dynamically updating the state of the simulation for a given timestep.*

any other proxies of interest can be used to design it. The glacier surface mass balance is downscaled (i.e. it depends) on S, which is updated by the solver, thus dynamically updating the state of the simulation for a given timestep.

In previous chapters we had described the Shallow Ice Approximation (SIA) equation we will use to model the flow of ice (Section 3.1.4). The fundamentals of universal differential equations and how they contextualize in the family of physics-informed machine learning methods in general was reviewed in Section 1.3.3. Finally, the differentiable programming machinery enabling gradient-descent optimization was introduced in Chapter 2 and broader developed in (Sapienza et al. 2024c). In this section we introduce how these elements are combined to give the universal differential equation associated to the SIA equation we will use to perform functional inversions of ice flow.

4.3.1 Forward model

In this study, we will focus in the SIA equation given by

$$\frac{\partial H}{\partial t} = \dot{b} + \nabla \cdot (D \nabla S), \qquad D = \left(C + \frac{2A}{n+2} H \right) (\rho g)^n H^{n+1} \|\nabla S\|^{n-1}, \tag{4.1}$$

where $H(x, y, t)$ is the length of the ice column; $\dot{b}$ is the mass balance (sink/source); $S(x, y, t) = B(x, y) + H(x, y, t)$ the ice surface profile; and the ∇ operator is with respect to the two spatial coordinates x and y (see Chapter 3). We consider a simple synthetic example where we fix Glen exponent $n = 3$, $C = 0$, and model the dependency of Glen's creep parameter A and the climate temperature normal T_s, i.e. the average long-term variability of air temperature at the glacier surface. Here T_s is computed using a 30-year rolling mean of the air temperature series, used to drive the changes in A in the prescribed artificial law. Although simplistic and incomplete, this relationship allows us to present all of our methodological steps in the process of identifying more general phenomenological laws for glacier dynamics. Any other proxies of interest could be used instead of T_s for the design of the model. Instead of considering that the diffusivity D_θ is the output of a universal approximator, we are going to replace the creep parameter A in Equation (3.25) with a neural network with input T_s, which results in the following functional version for the effective diffusivity

$$D_\theta(T_s) = \frac{2\, A_\theta(T_s)}{n+2} (\rho g)^n \, H^{n+2} \|\nabla S\|^{n-1}. \tag{4.2}$$

In this last equation, $D_\theta(T_s)$ is parametrized using a small neural network. The term $A_\theta(T_s)$ plays the role of β_θ in Equation (1.11), allowing the solutions of the PDE to span over a rich set of possible solutions. The objective of $A_\theta(T_s)$ will be to learn the spatial variability (i.e. among glaciers) of A with respect to T_s for multiple glaciers in different climates. However, this assumption ignores many other important physical drivers influencing the value of A, such as a direct relationship with the temperature of ice, the type of fabric (that is, the distribution of ice grain shape and orientation) and the water content (Cuffey et al. 2010). Nonetheless, this simple example serves to illustrate the modelling framework based on UDEs for glacier ice flow modelling, while acting as a platform to present both the technical challenges and adaptations performed in the process, and the future perspectives for applications at larger scales with additional data.

In order to solve the SIA Equation (3.23), we perform a discretization in the spatial domain to solve the problem as a combination of ordinary differential equations (see Section 3.2 for more details about how the numerical solver has been implemented). Once the problem has been discretized in the spatial domain, we use the numerical solver `RDPK3Sp35` (Ranocha et al. 2022) implemented in `DifferentialEquations.jl` (Rackauckas et al. 2016) to solve the SIA forward in time. The method implements an adaptive temporal step size close to the maximum value satisfying the CFL conditions for numerical stability at the same time that controls numerical error and computational storage (Ranocha et al. 2022).

As mentioned before, the function $A_\theta(T_s)$ is parametrized using a small feed-forward neural network using `Flux.jl` (Innes et al. 2018). The architecture of the neural network consists of one single input variable, 3 hidden layers with 3, 10, and 3 units, respectively, and one single output variable. Since the optimization problem is much more constrained by the structure of the solutions of the PDE compared to a pure data-driven approach (Rackauckas et al. 2020), a very small neural network is enough to learn the dynamics related to the subpart of the equation it is representing (i.e. A). This network has a single input and output neuron, thus producing a one-to-one mapping. This small size is one of the main advantages of UDEs, which do not require as much data as traditional data-driven machine learning approaches. We used a softplus activation function in all layers except for the last layer, for which we use a rescaled sigmoid activation function defined as

$$\sigma(x; x_{\min}, x_{\max}) = x_{\min} + \frac{x_{\max} - x_{\min}}{1 + e^{-x}} \tag{4.3}$$

which constrains the output within physically plausible values of $x_{\min} = 8^{-20} yr^{-1} Pa^{-3}$ to $x_{\max} = 8^{-17} yr^{-1} Pa^{-3}$. Constraining the output values of the neural network is necessary in order to avoid numerical instabilities in the solver or very small stepsizes in the forward model than will lead to expensive computations of the gradient. The use of smooth activation functions has been proven to be more effective for neural differential equations, since their derivatives are also smooth, thus avoiding problems of vanishing gradients (Kim et al. 2021).

4.3.2 Optimization and inverse model

Training an UDE requires that we optimize with respect to the solutions of the SIA equation, which need to be solved using numerical methods. The approach to fit the value of θ is to minimize the squared error between the target ice surface velocity profile (described in section 4.3.3 together with all other datasets used) at some given time and the predicted surface velocities using the UDE, an approach known as *trajectory matching* (Ramsay et al. 2017). For a single glacier, if we observed two different ice surface velocities u_0 and u_1 at times t_0 and t_1, respectively, then we want to find θ that minimizes the discrepancy between u_1 and $\text{SIASolver}(H_0, t_0, t_1, D_\theta)$, defined as the forward numerical solution of the SIA equation yielding a surface ice velocity field following Equation 3.26. When training with multiple glaciers, we are instead interested in minimizing the total level of agreement between observation and predictions,

$$\min_\theta \mathcal{L}(\theta) = \sum_k \omega_k \mathcal{L}_k(\theta), \qquad \mathcal{L}_k(\theta) = \|u_1^k - \text{SIASolver}(H_0^k, t_0, t_1, D_\theta)\|_F^2, \tag{4.4}$$

where $\|\cdot\|_F$ denotes the Frobenius norm; and each k corresponds to a different glacier. The weights ω_k are included in order to balance the contribution of each glacier to the total loss function. For our experiments, we consider $\omega_k = 1/\|u_0^k\|_F$, which results in a re-scaling of the surface velocities. This scaling is similar to the methods suggested in (Kim et al. 2021) to improve the training of stiff neural ODEs. In our case, the scaling also helps balancing the contribution of slow and fast flowing glacier to the total loss function.

Once the gradient of $\mathcal{L}(\theta)$ with respect to θ has been computed, optimization of the total loss function without any extra regularization penalty to the weights in the loss function was performed using a Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimizer with parameter 0.001. We also tested ADAM (Mogensen et al. 2018) with a learning rate of 0.01. BFGS converges in fewer epochs than ADAM, but it had a higher computational cost per epoch. Overall, BFGS performed better in different scenarios, resulting in a more reliable UDE training. For this toy model, a full epoch was trained in parallel using 17 workers, each one for a single glacier. Larger simulations will require batching either across glaciers, or across time in case a dataset with dense temporal series were used.

4.3.3 Training dataset

The following data are used for the initial and final glacier conditions:

1. A Digital Elevation Model (DEM) for the glacier surface elevation S based on the Shuttle Radar Topography Mission from the year 2005 (SRTM (Farr et al. 2007))

2. Estimated glacier ice thickness H from the global dataset from (Farinotti et al. 2019) based on

3. Glacier outlines around the year 2003 of the Randolph Glacier Inventory (Consortium 2017)

All these datasets, together with all glacier information are retrieved using the Open Global Glacier Model (OGGM), an open-source glacier evolution model in Python providing a vast set of tools to retrieve and process climate and topographical data related to glaciers (Maussion et al. 2019). Since these datasets are only available for just one or few times, but have a global spatial coverage of almost all of the ∼274,000 glaciers on Earth, we perform this training for 17 different glaciers distributed in different climates around the world (see Figure 4.3 and Table 4.1). This enables a good sampling of different climate conditions from which to take T_s to compute A. All climate data was based on the W5E5 climate dataset (Lange 2019), also retrieved using OGGM. For the purpose of the synthetic experiments, some of the boundary conditions (surface topography, glacier bedrock inferred from topography and ice thickness) are assumed to be perfectly known. Notice that the resolution of the spatial grid depends on the glacier size and domain size, typically ranging between 100x100 to 200x200 grid points, which leads to a system of coupled ODEs ranging from 10,000 to 40,000 equations per glacier.

In order to generate the forward simultion to train the UDE, we use a prescribed artificial law. For this we have used the relationship between ice temperature and A from (Cuffey et al. 2010) given in Table 4.2 and replaced ice temperatures with a relationship between A and T_s. This relationship is based on the hypothesis that T_s is a proxy of ice temperature,

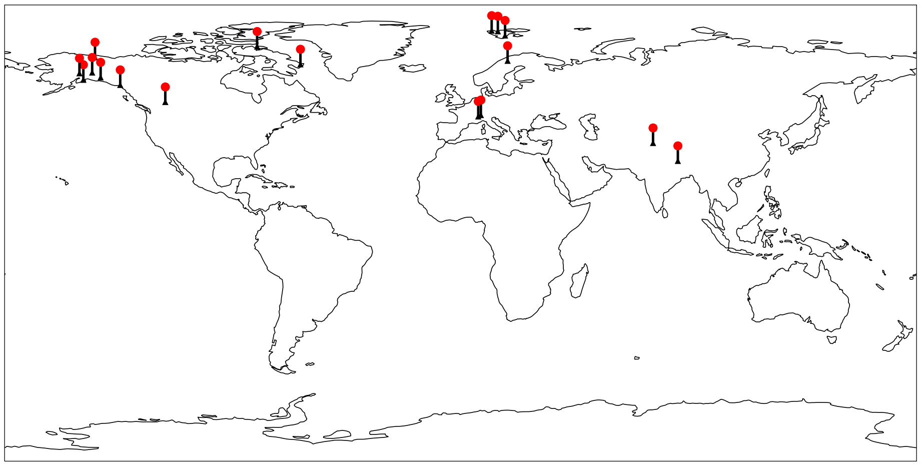


Figure 4.3: *Geographical distribution of the 17 glaciers used during training.*

and therefore of A. In order to create conditions similar to those one would encounter when using remote sensing observations for the functional inversions, we add Gaussian noise with zero mean and standard deviation $6 \cdot 10^{-18}$ (around 30% of the average A value) to the output of the prescribed law (Figure 4.5). This setup is used to compute the reference synthetic solutions (u_1^k in Equation (4.4)), which will then be used by the UDE to attempt to infer the prescribed law indirectly from glacier ice surface velocities and/or ice thickness.

4.3.3.1 Surface mass balance

In order to compute the glacier surface mass balance ($\dot{b}$ in Equations (3.2) and (4.1)) we used a very simple temperature-index model with a single melt factor and a precipitation factor set to 5 mm d^{-1} º C^{-1} and 1.2, respectively. These are average values found in the literature (Hock 2003), and despite its simplicity, this approach serves to add a realistic SMB signal on top of the ice rheology in order to assess the performance of the inversion method under realistic conditions. In order to add the surface mass balance term $\dot{b}$ in the SIA Equation (3.23) we used a `DiscreteCallback` from `DifferentialEquations.jl`. This enabled the modification of the glacier ice thickness H with any desired time intervals and without producing numerical instabilities when using all the numerical solvers available in the package (Rackauckas et al. 2016). We observed this makes the solution to be more stable without the need of reducing the stepsize of the solver. In order to find a good compromise between computational efficiency and memory usage, we pre-process raw climate files from W5E5 (Lange 2019) for the simulation period of each glacier. Then, within the run, for each desired timestep where the surface mass balance needs to be computed (monthly by

RGI ID	Glacier name	Region	Area (km^2)	Lon/Lat (º)	Grid size	Grid res (m)
RGI60-11.03638	Glacier d'Argentière	Central Europe	13.79	(6.98, 45.95)	(138, 129)	62
RGI60-11.01450	Aletschgletscher	Central Europe	82.2	(8.02, 46.50)	(107, 154)	137
RGI60-08.00213	Storglaciären	Scandinavia	3.40	(18.57, 67.90)	(110, 75)	36
RGI60-04.04351	-	Arctic Canada South	24.77	(-63.21, 66.52)	(132, 104)	80
RGI60-01.02170	Esetuk Glacier	Alaska	7.5	(-144.30, 69.29)	(138, 111)	48
RGI60-02.05098	Peyto Glacier	Western Canada and US	9.69	(-116.56, 51.65)	(104, 105)	54
RGI60-01.01104	Lemon Creek Glacier	Alaska	9.52	(-134.35, 58.38)	(75, 125)	53
RGI60-01.09162	Wolverine Glacier	Alaska	16.74	(-148.90, 60.41)	(96, 122)	67
RGI60-01.00570	Gulkana Glacier	Alaska	17.56	(-145.42, 63.28)	(132, 103)	69
RGI60-04.07051	-	Arctic Canada South	58.21	(-80.31, 73.52)	(102, 185)	117
RGI60-07.00274	Edvardbreen	Svalbard	61.18	(17.57, 77.88)	(132, 133)	120
RGI60-07.01323	Biskayerfonna	Svalbard	12.72	(12.28, 79.79)	(80, 122)	60
RGI60-01.17316	Twaharpies Glacier	Alaska	54.66	(-142.08, 61.36)	(195, 109)	114
RGI60-07.01193	Skaugumbreen	Svalbard	8.36	(14.72, 79.54)	(129, 116)	50
RGI60-01.22174	Buckskin Glacier	Alaska	46.46	(-150.45, 62.98)	(222, 93)	105
RGI60-14.07309	West Ching Kang Glacier	South Asia West	30.30	(75.9869, 35.4805)	(118, 135)	87
RGI60-15.10261	-	South Asia East	3.42	(85.788, 28.404)	(85, 138)	36

Table 4.1: *Table of glaciers used for training the UDE. Grid size and Grid res (i.e. resolution) indicate the adaptive constant grid used by OGGM to adapt all gridded data for each glacier.*

Ice temperature ($°C$)	$A(yr^{-1}Pa^{-3})$
0	7.57e-17
-2	5.36e-17
-5	2.93e-17
-10	1.10e-17
-15	6.63e-18
-20	3.79e-18
-25	2.15e-18
-30	1.17e-18
-35	6.31e-19
-40	3.16e-19
-45	1.64e-19
-50	8.20e-20

Table 4.2: *Recommended values of the Glen coefficient A as a function of the ice temperature. Creep coefficients A for the intermediate values of ice temperature are obtained with a polynomial interpolation (interpolation is shown in Figure 4.5). Table adapted from (Cuffey et al. 2010).*

default), we read the raw climate file for a given glacier and we downscale the air temperature to the current surface elevation S of the glacier given by the SIA PDE. For that, we use the adaptive lapse rates given by W5E5, thus capturing the topographical feedback of retreating glaciers in the surface mass balance signal (Bolibar et al. 2022).

4.3.4 Sensitivity methods and differentiation

In order to minimize the loss function from Equation (4.4), we need to evaluate its gradient with respect to the parameters of the neural network and then perform gradient-based optimization. Different methods exist to evaluate the sensitivity or gradients of the solution of a differential equation. These methods have been reviewed in Chapter 2. Here we compare the evaluation of the gradients using a continuous adjoint method integrated with automatic differentiation and a hybrid method that combines automatic differentiation with finite differences.

4.3.4.1 Continuous adjoint sensitivity analysis

For the first method based on pure automatic differentiation, we used the `SciMLSensitivity.jl` package, an evolution of the former `DiffEqFlux.jl` Julia package (Rackauckas et al. 2019), capable of weaving neural networks and differential equation solvers with AD. In order to train the SIA UDE from Equation (4.2), we use the same previously mentioned numerical scheme as for the PDE (i.e. `RDPK3Sp35`). Unlike in the original neural ODEs paper (Chen et al. 2018), simply reversing the ODE for the backward solve results in unstable gradients. This has been shown to be the case for most differential equations, particularly stiff ones like the one from this study (Kim et al. 2021). In order to overcome this issue, we used the interpolating adjoint method as described in Section 2.4.2.2.2. This method combined with an explicit solver proved more efficient than using the quadrature adjoint method. It is also possible to use checkpointing and just store a few evaluations of the solution. This has the advantage of reducing memory usage at the cost of sacrificing some computational performance (Griewank et al. 2008; Schanen et al. 2023). To compute the vector-Jacobian products involved in the adjoint calculations of the SIA UDE, we used reverse-mode AD with the `ReverseDiff.jl` package with a cached compilation of the reverse tape. We found that for our problem, the limitation of not being able to use control flow was easily bypassed, and performance was noticeably faster than other AD packages in Julia, such as `Zygote.jl`.

4.3.4.2 Finite differences

The second method consists in using AD just for the neural network and finite differences for capturing the variability of the loss function with respect to the parameter A. Notice that in Equation (4.4) we can write $\mathcal{L}_k(\theta) = \mathcal{L}_k(A_\theta(T_k))$, with $A(\theta)$ the function that maps input parameters T_k into the scalar value of A (which for this example is assumed to be a single scalar across the glacier) as a function of the neural network parameters θ. Once A has being specified, the function $\mathcal{L}_k$ is a one-to-one function that is easily differentiable using finite differences. If we define $g = \nabla_\theta A(T_k)$ the gradient of the neural network that we obtain using AD, then the full gradient of the loss function can be evaluated using the centered numerical approximation

$$\nabla_\theta \mathcal{L}_k \approx \frac{\mathcal{L}(\theta + \eta g) - \mathcal{L}(\theta - \eta g)}{2\eta \|g\|^2}\, g, \quad g = \nabla_\theta A_\theta(T_k), \tag{4.5}$$

where η is the stepsize used for the numerical approximation of the derivative. Notice that the first term on the right hand side is just a scalar that quantifies the sign and amplitude of the gradient, which will be always in the direction of $\nabla_\theta A(T_k)$. The choice of stepsize η is critical in order to correctly estimate the gradient. Notice that this method works just when there are a few parameters, and will not generalize well to the case of an A that varies in space and time for each glacier. The main advantage of this method is that it very simple to implement and it does not require the calculation of the continuous adjoint.

4.3.5 Scientific computing in the future

As part of this new approach in terms of geoscientific computing, we are computing everything directly in the cloud using a JupyterHub (see `https://jupyter.org/hub`). This JupyterHub allows us to work with both Julia and Python, using Unix terminals, Jupyter

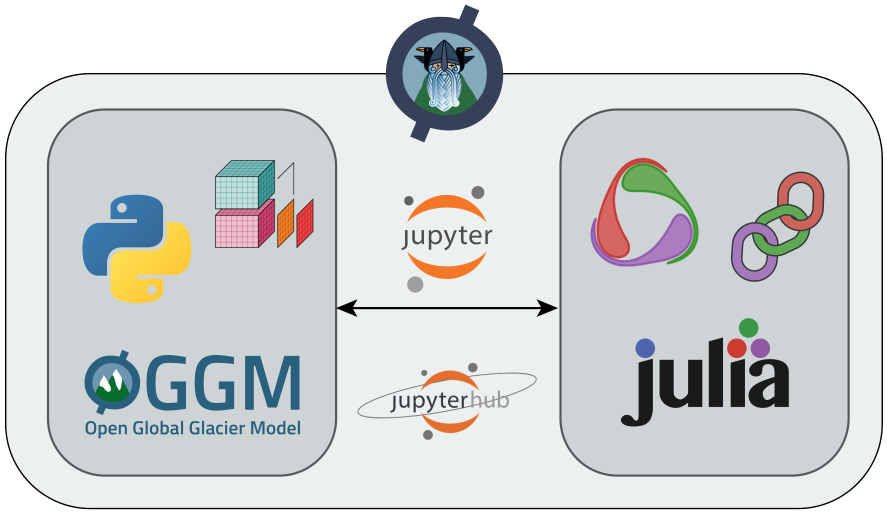


Figure 4.4: *Multilanguage framework inside* `ODINN.jl`. *Most of the source code is written in Julia, but the preprocessing of the glacier data is performed using the Open Global Glacier Model (OGGM) and* `xarray` *in Python.*

notebooks (Kluyver et al. 2016) and VSCode directly on the browser. Moreover, this provides a lot of flexibility in terms of computational resources. When logging in, one can choose between different machines, ranging from small ones (1-4 CPUs, 4-16 GB RAM) to very large ones (64 CPUs, 1 T4 Tensor Core GPU, 1 TB RAM), depending on the task to be run. The largest simulations for this study were run in a large machine, with 16 CPUs and 64 GB of RAM.

ODINN is a multilanguage library that is mostly written in Julia but run sub-routines in Python. This is done by the use of the package `PyCall.jl`, which enables a seamless integration of Python libraries such as OGGM and `xarray` (Hoyer et al. 2017b) within ODINN. Figure 4.4 shows an schematic of the different software tools present in the ODINN ecosystem.

4.4 Results

Despite its apparent simplicity, it is not a straightforward problem to invert the spatial function of A with respect to predictor indirectly from surface velocities, mainly due to the highly nonlinear behaviour of the diffusivity of ice (see Figure 3.2). We ran a functional inversion using two different differentiation methods for 17 different glaciers (Table 4.1) for a period of 5 years.

Training the UDE with full batches using the continuous adjoint method converges in around 20 epochs. The neural network is capable of successfully approximating the pre-

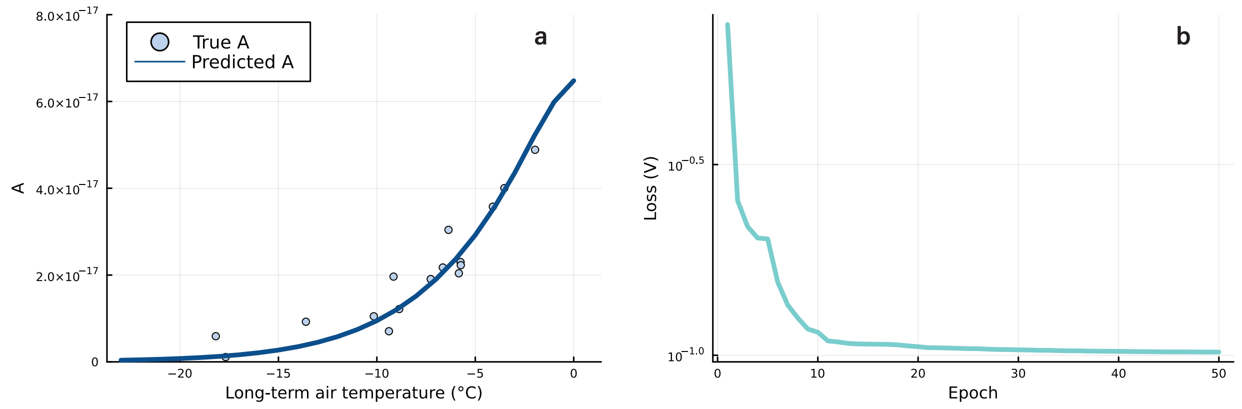


Figure 4.5: *(a) Inferred function by the neural network embedded in the SIA PDE using full automatic differentiation. The neural network learnt the prescribed noisy function (each dot represents a glacier), that relates Glen's coefficient A with a proxy of interest (i.e. the long-term air temperature T_s). (b) Evolution of the loss through training, using a BFGS optimizer. The loss is based on a scaled mean squared error of the difference between the simulated and target ice surface velocities. The scaling is used to correctly account for values of different orders of magnitude. Without any use of regularization, the optimization converges in around 20 epochs. Note that the loss is shown in log scale.*

scribed nonlinear function of A. The loss sees a steep decrease in the first epochs, with BFGS optimizing the function towards the lowest values of A, which correctly approximate the majority of values of the prescribed nonlinear function. From this point, the function slowly converges until it finds an optimal non-overfitted solution (Figure 4.5b). This simulation took about 3 hours to converge, with a running time between 6 to 12 minutes per epoch, in a machine in the cloud-based JupyterHub with 16 CPUs and 64 GB of RAM, using all available CPUs to simulate in parallel the 17 glaciers in batches and using the full 64 GB of memory. Figure 4.5a shows the parametrization of A as a function of T_s obtained with the trained neural network. We observe that the neural network is able to capture the monotonic increasing function $A(T_s)$ without overfitting the noisy values used for training (dots in the plot). Interestingly, the lack of regularization did not affect overfitting. We are unsure about the reasons behind this behaviour, but we suspect this could be related to an implicit regularization caused by UDEs. This can be related to the shape of the landscape obtained by the map forward of the solutions of the differential equation, which makes more likely to find local minima that result in smooth functions $A_\theta(T_s)$ instead of overfitted function with lower empirical error. In this sense, a bold speculation is that the differential equation serves as an implicit regularizer of the objective function. This property has not been studied yet, so more investigations should be carried out in order to better understand this apparent robustness to overfitting.

We also compared the efficiency of our approach when using the finite differences scheme.

Since this does not require heavy backwards operations as the continuous adjoint method does, the finite difference method runs faster (around 1 minute per epoch). However, we encountered difficulties in picking the right stepsize η in Equation (4.5). Small values of η lead to floating number arithmetic errors and large η to biased estimates of the gradient. On top of this effect, we found that numerical precision errors in the solution of the differential equation result in wrong gradients and the consequent failure in the optimization procedure (see discussion about this in Section 4.5.2.3). A solution for this would be to pick η adaptive as we update the value of the parameter θ. However, this would lead to more evaluations of the loss function. Instead, we applied early stopping when we observed that the loss function reached a reasonable minimum.

4.4.1 Robustness to noise in observations

The addition of surface mass balance to the SIA equation further complicates things for the functional inversion, particularly from a computational point of view. The accumulation and ablation (subtraction) of mass on the glacier introduces additional noise to the pure ice flow signal. The mass loss in the lower elevations of the glacier slows down ice flow near the tongue, whereas the accumulation of mass in the higher elevations accelerates the ice flow on the upper parts of the glacier.

As an experiment to test the robustness of the functional inversions made by the UDE, we used different surface mass balance models for the reference simulation (i.e. the ground truth), and the UDE. This means that the surface mass balance signal during training is totally different from the one in the ground truth. We achieved this by using a temperature-index model with a melt factor of 4 mm d^{-1} º C^{-1} and a precipitation factor of 2 for the reference run, and a melt factor of 8 mm d^{-1} º C^{-1} and a precipitation factor of 1 for the UDE. This means that the UDE is perceiving a much more negative surface mass balance than the actual one from the ground truth. Despite the really large difference that can be seen in Figure 4.6, the UDE was perfectly capable of inverting the nonlinear function of A. The evolution of the loss was less smooth than for the case of matching surface mass balance rates, but it also converged in around 20 epochs, with no noticeable difference in final performance.

This shows the robustness of this modelling approach, particularly when the ice surface velocities u are used in the loss function. Unlike the glacier ice thickness, u is much less sensitive to changes in surface elevation, making it a perfect data for inferring ice rheology properties. This is also due to the fact that we are using ice surface velocities averaged across multiple years, which lessen the importance of changes in surface elevation. When working with velocities with a higher temporal resolution, these will likely become more sensitive to noise. This weak dependence on the surface mass balance signal will be highly beneficial for many applications, since it will imply that the inversion can be done even if we only have an approximate representation of the surface mass balance, which will be the case for many glaciers. Various tests using H instead of u as the target data showed that A cannot be successfully inverted in the presence of a surface mass balance signal.

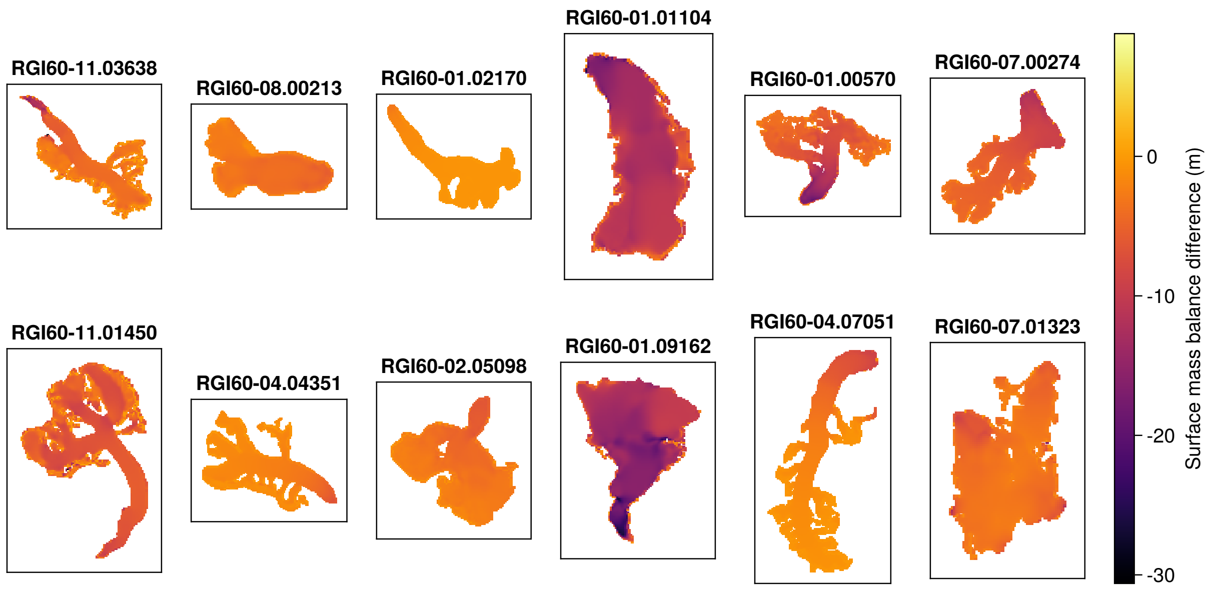


Figure 4.6: *Differences in surface elevation for a 5-year simulation, coming from the different applied surface mass balance rates, between the ground truth data and the training of the UDE. Despite the noise coming from the different surface mass balance signal, the UDE is perfectly capable of learning the underlying nonlinear function of A. This proves the robustness against noise of this functional inversion framework for glacier rheology when using ice surface velocities. Showing only 12 out of the total 17 glaciers.*

4.5 Discussion: challenges and perspectives

4.5.1 Application to functional inversions of glacier physical processes

This first implementation of a UDE on glacier ice flow modelling serves as a baseline to tackle more complex problems with large datasets. One main simplification of this current setup needs to be overcome in order to make the model useful at a global scale for exploring and discovering empirical laws. In this study, only ice deformation (creep) has been taken into account in the diffusivity. Basal sliding, at the ice-bedrock interface, will have to be included in the SIA equation to accommodate different configurations and behaviours of many glaciers around the world. Therefore, a logical next step would be to infer D in Equation (3.25), including the sliding coefficient C from Equation (3.23) using a UDE. Nonetheless, despite a scale difference between these two processes, this can be an ill-posed problem, since the only available ice velocity observations are from the surface, encompassing both creep and basal sliding. This results in degeneracy, making it very challenging to disentangle the contributions of each physical processes to ice flow. This is particularly true for datasets with average annual surface velocities, since both physical processes contribute to glacier velocities, with no obvious way of separating them. In order to overcome such issues, using new remote sensing datasets with high temporal resolution, like (Nanni et al. 2023) with observations every 10 days, can help better constrain the contribution of each physical process. This implies that we cannot only exploit the spatial dimension with multiple glaciers, but also rich time series of the fluctuations of glacier velocities along the seasons. Such dynamics can help disentangle the main contributions of creep during the winter season, and the onset of sliding during the summer season as the subglacial hydrological network activates due to melt.

Interestingly, depending on the used ice surface velocity observations, the need of a numerical solver and a UDE are not imperative for a functional inversion. For a single snapshot of ice surface velocities between two dates (e.g. 2017-2018 in (Millan et al. 2022)), a functional inversion can be performed directly on the SIA equation without the need of a solver. The average ice surface velocities can be directly inverted if the geometry is known. This reduces the technical complexity enormously, enabling one to focus on more complex neural network architectures and functions to inverse ice rheology and basal sliding properties. Some initial tests have shown that such problems train orders of magnitude faster. However, since only one timestamp is present for the inversions, the inversion is extremely sensitive to time discrepancies in the input datasets, making it potentially quite vulnerable to noisy or temporally misaligned datasets.

Alternatively, the optimization of the neural network for ice rheology inference based on ice surface velocities has proved to be robust to the noise added by the surface mass balance component. This serves to validate an alternative glacier ice dynamics model calibration strategy to those of the majority of large-scale glacier models (e.g. OGGM and GloGEM; (Huss et al. 2015)). By being able to calibrate separately ice rheology and mass balance, one can avoid many equifinality problems that appear when attempting to calibrate both components at the same time (Arthern et al. 2010; Zhao et al. 2018). A classic problem of a joint calibration is the ambiguity in increasing/decreasing accumulation vs increasing/decreasing Glen's coefficient (A). `ODINN.jl`, with its fully differentiable codebase, provides a different strategy consisting in two main steps: (i) calibrating the ice rheology from observed ice surface velocities (Millan et al. 2022), observed ice thicknesses (Consortium 2019) and DEMs; (ii) calibrating the MB component (e.g. melt and accumulation factors) with the previously calibrated ice rheology based on both point glaciological mass balance observations and multiannual geodetic MB observations (Hugonnet et al. 2020). This maximises the use of current glaciological datasets for model calibration, even for transient simulations. Such a differentiable modelling framework presents both the advantages of complex inversions and the possibility of performing data assimilation with heterogeneous and sparse observations.

4.5.2 Scientific machine learning

4.5.2.1 Automatic differentiation approaches

Automatic differentiation is a centerpiece of the modelling framework presented in this study. In the Julia programming language, multiple AD packages exist, which are compatible with both differential equation and neural networks packages, as part of the SciML ecosystem. Each package has advantages and disadvantages, which make them suitable for different tasks. In our case, `ReverseDiff.jl` turned out to be the best performing AD package, due to the speed gained by reverse tape compilation. Together with `Zygote.jl` (Innes et al. 2019), another popular reverse AD package, they have the limitation of not allowing mutation of arrays. This implies that no in-place operations can be used, thus augmenting the memory allocation of the simulations considerably. `Enzyme.jl` (Moses et al. 2021) is arising as a promising alternative, with the fastest gradient computation times in Julia (Ma et al. 2021b). It directly computes gradients on statically analyzable LLVM, without differentiating any Julia source code. Nonetheless, `Enzyme.jl` is still under heavy development, and it is still not stable enough for problems like the ones from this study. As `Enzyme.jl` will become more robust in the future, appears likely to become the *de facto* solution for AD in Julia.

Overall, the vision on AD from Julia is highly ambitious, attempting to perform AD directly on source code, with minimal impact on the user side and with the possibility of easily switching AD back-ends. In practice, this implies that it is much more complex to achieve from a technical point of view than hardcoded gradients linked to specific operators, an approach followed by JAX (Bradbury et al. 2020) and other popular deep learning Python libraries. On the short term, the latter provides a more stable experience, albeit a more rigid one. However, in the long term, once these packages are given the time to grow and become more stable, differentiating through complex code, like the one from UDEs, should become increasingly straightforward.

4.5.2.2 Surrogate models and emulators

In this work, we model glacier ice flow using a two-dimensional SIA equation described by Equation (3.23). This decision was originally driven by the goal of using a differential equation that is general enough for modelling the gridded remote sensing available data, but also flexible enough to include unknowns in the equation in both the diffusivity and surface mass balance terms. Nonetheless, the approach of UDEs and functional inversions is flexible, and can be applied to more complex ice flow models, such as Full-Stokes. It is important to keep in mind that for such more complex models, the numerical solvers involved would be computationally more expensive to run, both in forward and reverse mode.

A recent alternative to such a computationally heavy approach is the use of surrogate models and emulators (see Section 1.3.1). An example of these are the emulators based on convolutional neural networks for the solutions of the differential equations of a high-order ice flow model (Jouvet et al. 2021; Wang et al. 2022). Emulators can be used to replace the numerical solver shown in the workflow illustrated in Figure 4.2, while keeping the functional inversion methodology intact. This in fact translates into replacing a numerical solver by a previously trained neural network, which has learnt to solve the Stokes equations for ice flow. Embedding a neural network inside a second neural network that gives approximate solutions instead of a PDE solved by a numerical solver allows computing the full gradient of the loss function by just using reverse AD. Neural networks can be very easily differentiated, thus resulting in a simpler differentiation scheme. At the same time, as shown in (Jouvet 2023), this could potentially speed-up functional inversions by orders of magnitude, while also improving the quality of the ice flow simulations transitioning from the SIA, with its vertically-integrated ice column assumption, to Full-Stokes.

4.5.2.3 New statistical questions

The combination of solvers for differential equations with modern machine learning techniques opens the door to new methodological questions that include the standard ones about the design of the machine learning method (loss function, optimization method, regularization) but also new phenomena that emerges purely by the use of numerical solutions of differential equations in the loss function. Although this intersection between data science and dynamical systems has been widely explored (see (Ramsay et al. 2017)), the use of adjoints for sensitivity analysis integrated with AD tools for optimization and the properties of the landscape generated when using numerical solvers has not. Because numerical methods approximate the solution of the differential equation, there is an error term associated to running the forward model that can be amplified when running the model backwards when computing the adjoint. This can lead to inaccurate gradients, especially for stiff differential equations (Kim et al. 2021). Furthermore, when computing a loss function that depends of the solution of the differential equation for certain parameter θ, the loss depends on θ because local variations in θ result in changes in the error itself, but also because of hyper-parameters in the numerical solver (for example, the timestep used to ensure stability) that are adaptatively selected as a function of θ. This double dependency of the loss as a function of θ results in distortions of the shape of the loss function (Creswell et al. 2023). Further investigation is needed in order to establish the effect of these distortions during optimization and how these can impact the calculation of the gradient obtained using different sensitivity methods.

Another interesting question regards the training and regularization of UDEs and related physics-informed neural networks. During training, we observed that the neural network never overfitted the noisy version of prescribed law $A(T_s)$. We conjecture that one reason why this may be happening is because of the implicit regularization imposed by the form of the differential equation in Equation (3.23).

4.6 Conclusions and future directions

Despite the ever increasing amounts of new Earth observations coming from remote sensing, it is still extremely challenging to translate complex, sparse, noisy data into actual models and physical laws. Paraphrasing (Rackauckas et al. 2020), "In the context of science, the well-known adage *a picture is worth a thousand words* might well be *a model is worth a thousand datasets".* Therefore, there is a clear need for new modelling frameworks capable of generating data-driven models with the interpretability and hard constraints of classic physical models. Universal differential equations (UDEs) embed a universal function approximator (e.g. a neural network) inside a differential equation. This enables additional flexibility typical from data-driven models into a reliable physical structure determined by a differential equation.

We presented `ODINN.jl`, a new modelling framework based on UDEs applied to glacier ice flow modelling. We illustrated how UDEs, supported by differentiable programming in the Julia programming language, can be used to retrieve empirical laws present in datasets, even in the presence of noise. We did so by using the Shallow Ice Approximation PDE, and learning a prescribed artificial law as a subpart of the equation. We used a neural network to infer Glen's coefficient A, determining the ice viscosity, with respect to a climatic proxy for 17 different glaciers across the world. The presented functional inversion framework is robust to noise present in input observations, particularly on the surface mass balance, as shown in an experiment.

This study can serve as a baseline for other researchers interested in applying UDEs to similar nonlinear diffusivity problems. It also provides a codebase to be used as a backbone to explore new parametrizations for large-scale glacier modelling, such as for glacier ice rheology, basal sliding, or more complex hybrid surface mass balance models.

Software availability. The source code of ODINN.jl v0.2.0 (Bolibar et al. 2023b) used in this study is available as an open-source Julia package: `https://github.com/ODINN-SciML/ODINN.jl`. The package includes Continuous Integration tests, installation guidelines on how to use the model and a Zenodo DOI: `https://zenodo.org/record/8033313`. OGGM v1.6 (Maussion et al. 2023) is also available as an open-source Python package at: `https://github.com/OGGM/oggm`, with documentation and tutorials available at `https://oggm.org`.

Chapter 5

Quantitative analysis of paleomagnetic sampling strategies

A different application of statistical modelling in geophysics is pursued in the field of Paleomagnetism. Most of the contents of this chapter are based on the analysis and results in

- F. Sapienza, L. C. Gallo, Y. Zhang, B. Vaes, M. Domeier, and N. L. Swanson-Hysell (2023a). "Quantitative Analysis of Paleomagnetic Sampling Strategies". In: *Journal of Geophysical Research: Solid Earth* 128.11, e2023JB027211. DOI: https://doi.org/10.1029/2023JB027211.

The associated publication includes an editor highlight in the *Journal of Geophysical Research: Solid Earth* titled *Should I Stay or Should I Go... To Another Paleomagnetic Site?* available at https://eos.org/editor-highlights/should-i-stay-or-should-i-goto-another-paleomagnetic-site. At the end of this chapter we will further show how the same tools covered in Chapter 4 can be used to model the past motion of tectonic plates from paleomagnetic data. Further work in estimation of apparent polar wander paths (see Section 5.7.1) can be found in the following publications:

- L. C. Gallo, M. Domeier, F. Sapienza, N. L. Swanson-Hysell, B. Vaes, Y. Zhang, M. Arnould, A. Eyster, D. Gürer, Á. Király, B. Robert, T. Rolf, G. Shephard, and A. van der Boon (2023). "Embracing Uncertainty to Resolve Polar Wander: A Case Study of Cenozoic North America". In: *Geophysical Research Letters* 50.11. DOI: 10.1029/2023gl103436
- L. C. Gallo, F. Sapienza, and M. Domeier (2022). "An optimization method for paleomagnetic Euler pole analysis". In: *Computers & Geosciences* 166, p. 105150

and in the two manuscripts in progress

- F. Sapienza et al. (2024b). "Fitting curves in the sphere using universal differential equations". In: *preparation*

- L. C. Gallo et al. (2024). “On the feasibility of paleomagnetic Euler pole analysis”. In: *preparation*

5.1 Abstract

Sampling strategies used in paleomagnetic studies play a crucial role in dictating the accuracy of our estimates of properties of the ancient geomagnetic field. However, there has been little quantitative analysis of optimal paleomagnetic sampling strategies and the community has instead defaulted to traditional practices that vary between laboratories. In this paper, we quantitatively evaluate the accuracy of alternative paleomagnetic sampling strategies through numerical experiments and an associated analytical framework. Our findings demonstrate a strong correspondence between the accuracy of an estimated paleopole position and the number of sites or independent readings of the time-varying paleomagnetic field, whereas larger numbers of in-site samples have a dwindling effect. This remains true even when a large proportion of the sample directions are spurious. This approach can be readily achieved in sedimentary sequences by distributing samples stratigraphically, considering each sample as an individual site. However, where the number of potential independent sites is inherently limited the collection of additional in-site samples can improve the accuracy of the paleopole estimate (although with diminishing returns with increasing samples per site). Where an estimate of the magnitude of paleosecular variation is sought, multiple in-site samples should be taken, but the optimal number is dependent on the expected fraction of outliers. The use of filters based on angular distance helps the accuracy of paleopole estimation, but leads to inaccurate estimates of paleosecular variation. We provide both analytical formulas and a series of interactive Jupyter notebooks allowing optimal sampling strategies to be developed from user-informed expectations.

Plain Language Summary

Earth’s magnetic field can be preserved in rocks when they form. Through studying these magnetic records using the tools of paleomagnetism, scientists can learn about how Earth’s magnetic field has changed through time and how tectonic plates have moved relative to the field. This study is about the best ways to design sampling approaches to gain these insights using statistical quantification. Traditional protocols emphasize the collection of numerous samples from units that record the field at a given instant in time. Such units are referred to as sites. By simulating data, we develop tools for evaluating trade offs between collecting more sites and more samples per site. Our results show that strategies that maximize collecting more sites, even if fewer samples are taken at each site, leads to more accurate estimates even in the presence of spurious observation. While there is a benefit to more samples per site, particularly for studies seeking to estimate the variability of the ancient field, such sampling has diminishing returns relative to maximizing the number of sites. We

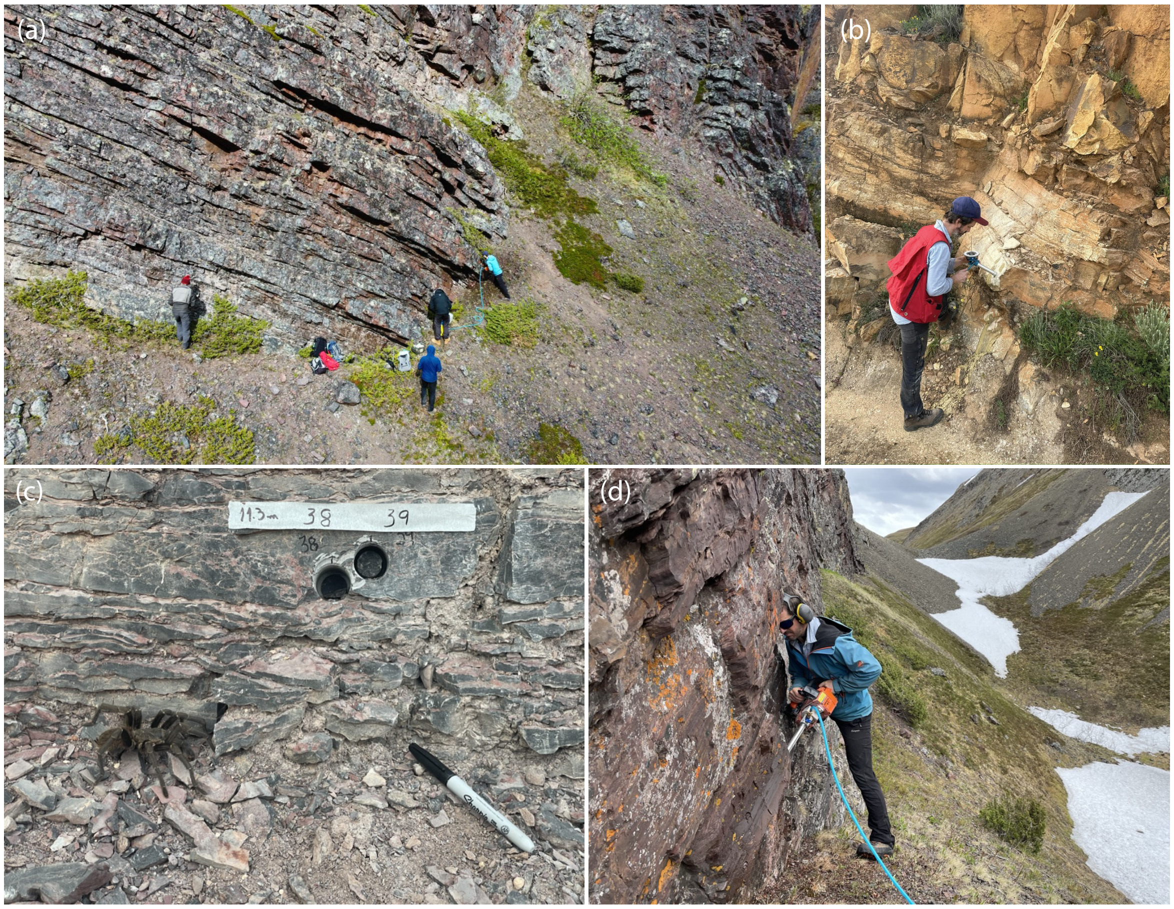


Figure 5.1: *A paleomagnetic campaign. Samples consist in cylindrical pieces of rock (c) extracted from a given outcrop (a). Different layers are associated to different ages and sites. Pictures (b) and (d) despite Mathew Domeier measuring the orientation and drilling the new sample, respectively. Please notice that in all this chapter, samples correspond to rock samples, no statistical samples. Pictures courtesy of Leandro Gallo.*

provide formulas and interactive computational resources to help the community to make informed decisions about the best way to gather data.

5.2 Introduction

Paleomagnetism is concerned with attempting to estimate properties of the ancient geomagnetic field from magnetic records preserved in rocks. This involves laboratory measurements of magnetization directions recorded by rocks and statistical analyses of those directions. Two geomagnetic properties of particular interest that can be estimated from these paleomagnetic directional data are:

- The position of the time-averaged ($\gtrsim 10^4 - 10^5$ years) ancient geomagnetic pole (also known as a *paleopole*) that corresponds to the Earth's spin axis according to the geocentric axial dipole hypothesis (Creer et al. 1954).
- The paleosecular variation of the field, which is associated with the shorter-term ($\lesssim 10^4 - 10^5$ years) time-varying position of the geomagnetic pole.

Despite the importance of these two quantities, there has been little exploration of the best sampling practices with which to derive estimates of them. This has resulted in practices that vary according to the traditions of different laboratories; that is, the community largely relies on conventional wisdom.

In the hierarchical framework of paleomagnetic studies, a site should correspond to a unit of rock with a common age and direction of magnetization (McElhinny et al. 2000; Tauxe 2010). Note that in some contributions a site is defined more loosely as a small area or stratigraphic interval from which samples are collected which is not the definition that we use here. In our preferred definition, each site is interpreted to be a *spot recording* of the time-varying geomagnetic field. In the case of an igneous rock, a site could be an individual lava flow or intrusion, whereas for a sedimentary rock, a site should ideally comprise a single depositional event. In practice, a sedimentary site typically corresponds to a single stratigraphic horizon that is the height of a standard paleomagnetic sample, usually about 2.5 cm (see Figure 5.1). Notice that when sedimentation rates are low, an individual samples may partially time average the field. To move up the hierarchy, a collection of paleomagnetic samples from a given site are averaged and the site mean is transformed from a direction with an associated declination and inclination to pole space with an associated latitude and longitude, where the mean is referred to as a *virtual geomagnetic pole* (VGP). Following the definition of a site, each VGP ideally represents an independent estimate of the position of the ancient geomagnetic pole at an instant in time. Estimates of paleosecular variation of the ancient geomagnetic field prior to 10 Ma can be made from populations of VGPs by determining their angular dispersion – most typically applied to collections of igneous sites of a similar age (McFadden et al. 1988). To determine a mean paleomagnetic pole position, a group of similarly aged VGPs are averaged to a Fisher mean paleopole that is taken as the best estimate of the true position of the ancient geographic pole relative to the observation point.

Regardless of whether we seek to discern the statistical properties of the time-averaged pole position or geomagnetic secular variation, our estimates will include error. Paleomagnetic errors come from a variety of sources which can include orientation errors both in the field and the laboratory; measurement errors; and the imperfect isolation of the magnetization of interest from secondary magnetic overprints. The frequent occurrence of imperfect magnetization acquisition or the inability to isolate primary components often results in a sample collection being contaminated by outliers. Orientation and measurement errors are generally assumed to be randomly unbiased (non-systematic) and so can be mitigated through the collection, measurement and directional averaging of multiple samples within a site. However, given finite resources, the collection of additional samples per site will come at the cost of a lower number of sites in many settings. A relevant question is thus: how should we distribute our sampling to minimize uncertainty on the property we seek to estimate? Is it better to take a few sites with many samples? Or many sites with fewer samples? How might the recommended strategy change depending on the objective (in estimating the location of the paleopole vs. the dispersion of VGPs) or the fidelity of the magnetic record?

Some notions concerning sampling have become entrenched in the paleomagnetic literature. For example, many workers seek to collect six to eight samples per site (Butler 1992), although the rationale for this range is not entirely clear. (Opdyke et al. 1996) suggest that at least three samples per site be collected where determinations of polarity are important, whereas to reliably estimate the dispersion of sample directions within a site, a minimum of four (Cromwell et al. 2018) or five (Tauxe et al. 2003) samples per site has been deemed necessary. Having a more significant number of samples within the site provides the benefit of being able to apply data filters based on within-site scatter. However, (Gerritsen et al. 2022) have found empirically that collecting and averaging multiple samples per site only results in a modest enhancement of the overall accuracy of the paleopole. Thus, where the objective is to estimate the position of a paleopole, (Gerritsen et al. 2022) suggested that it is most beneficial to maximize the number of sites, and so the collection of additional single-sample sites should be preferred over the collection of multiple samples from fewer sites. Nevertheless, a statistical and quantitative evaluation of alternative strategies has not yet been conducted.

Here we explore how the distribution of samples across sites affects the performance in the estimation of the paleopole position and the dispersion of VGPs, and how the varying influence of outliers dictates the optimal strategy to best estimate these parameters. We also derive a set of equations that can enable quantitative sampling strategy recommendations based on specified parameters informed by user expectations.

5.3 Mathematical setup

Consider the problem of estimating a paleomagnetic pole μ_0 for some given interval of time, where μ_0 is a three-dimension vector contained in the unit sphere. Observations consist of a collection of a total of n samples distributed among N sites. Because the geomagnetic field is constantly varying around a mean configuration, each one of the virtual geomagnetic poles (VGP) per site, denoted by μ_i with $i = 1, 2, \dots, N$, is going to differ from the time-averaged paleomagnetic pole μ_0. A fundamental assumption in paleomagnetic research is that this secular variation of the geomagnetic field can be effectively estimated through averaging of a sufficiently high number of independent and temporally distributed VGPs. We now seek to evaluate how our choices of n and N will affect our estimation of μ_0, as well as how we distribute the n samples among the N sites.

5.3.1 Data generating process

We define the following data generating process. First, we consider a set with a total of N VGPs sampled from a statistical model of secular variation. Examples of these models include the Gaussian process type model (Constable et al. 1988; Tauxe et al. 2004) and model G (McFadden et al. 1988). In this contribution, we use model G which captures latitudinal variation in VGP scatter, and considers a mean geocentric axial and dipolar (GAD) field. Then, given a GAD mean direction μ_0, we sample a series of VGPs $\mu_1, \mu_2, \dots, \mu_N$ according to

$$\mu_i \sim \text{SV}(\mu_0, \kappa_b) \quad i = 1, 2, \dots, N. \tag{5.1}$$

The sampling procedure depends on the mean direction μ_0 and the precision parameter κ_b that will depend on the secular variation model used. In this study, we adopt the mild assumption that VGP distributions are circularly symmetric (Tauxe et al. 2004) and can be sampled from a Fisher distribution (Deenen et al. 2011; Fisher 1953), whose dispersion S_b, according to model G (McFadden et al. 1988), depends on the sampling latitude λ through the following formula

$$S_b(\lambda)^2 = a^2 + b^2\lambda^2, \tag{5.2}$$

with a and b two empirical coefficients, recently calculated as $a = 11.3^{\circ}{}^{+1.3^{\circ}}_{-1.1^{\circ}}$ and $b = 0.27^{+0.04}_{-0.08}$ by (Doubrovine et al. 2019). At population level, there is a one-to-one relationship between S_b and the value of κ_b we use to sample from the Fisher distribution. This relationship can be found numerically with an arbitrary level of precision. Then, VGPs can be sampled according to a Fisher distribution with mean direction μ_0 and dispersion parameter $\kappa_b(\lambda)$.

In the following, we use the supraindex $*$ to denote variables in directional space (inclination-declination). Thus, μ_i refers to any given VGP (geographic coordinates) and μ_i^* refers to its corresponding direction in inclination and declination space according to the dipole formula. Note that this transformation between pole and directional space depends on the latitude and longitude of the site.

Now, we assume that the ith-site has n_i individual directions that follow a Fisher distribution

$$\begin{aligned} x_{ij}^* &\sim \text{Fisher}(\mu_i^*, \kappa_i) \quad \text{with probability } 1 - p_{\text{outlier}} \text{ and} \\ x_{ij}^* &\sim \text{Unif} \quad \text{otherwise, for } j = 1, 2, \dots, n_i, \end{aligned} \tag{5.3}$$

with x_{ij} the jth-direction of the ith-site; κ_i the dispersion parameters per site; and Unif represents the uniform distribution on the sphere. The parameter p_{outlier} has been added to quantify the effect of outliers in the sampling process. With probability $1 - p_{\text{outlier}}$ we are going to observe a true sample, while with probability p_{outlier} our sample will be corrupted and instead we will observe a spurious direction, modelled by a uniform distribution on the sphere where no information is provided about the true orientation of the field. For cases where we do not want to consider the effect of outliers in the sampling process, we set $p_{\text{outlier}} = 0$. Also, for cases where the number of samples and dispersion parameter are the same for all the sites, we will use n_0 and κ_w to refer to any of the n_i and κ_i, respectively. The parameters used in the model are summarized in Table 5.1.

Parameter	Range	Description
N	≥ 1	Total number of sites.
n_0	≥ 1	Number of samples per site. We will assume $n_0 = n_1 = \ldots = n_N$ and denote $n = Nn_0$ the total number of samples.
κ_w	$[0, \infty)$	Precision parameter of the Fisher distribution for a given site, where $k_w = 0$ results in a uniform distribution on a sphere and $k_w \to \infty$ is a singular point.
κ_b	$[0, \infty)$	Precision parameter of the Fisher distribution between sites. For the model G, this is directly determined by λ.
λ	$[0°, 90°]$	Paleolatitude.
p_{outlier}	$[0, 1]$	Outlier rate where 0 is no outliers and 1 is all samples are outliers drawn from a uniform distribution.

Table 5.1: *Parameters used in the data generating process for the sampling of poles.*

5.3.2 Estimation of the paleopole direction

We can estimate the true pole location μ_0 by computing the Fisher mean of the VGPs estimated from each site, that is,

$$\hat{\mu}_0 = \frac{1}{R_0} \sum_{i=1}^{N} \hat{\mu}_i \quad R_0 = \left\| \sum_{i=1}^{N} \hat{\mu}_i \right\|, \tag{5.4}$$

where R_0 is the length of the resultant vector with $\| \cdot \|$ denoting the Euclidean norm; and $\hat{\mu}_i$ is the sample mean per site, which results from transforming to pole space the estimate of the pole in directional space,

$$\hat{\mu}_i^* = \frac{1}{R_i} \sum_{j=1}^{n_i} x_{ij}^* \quad R_i = \left\| \sum_{j=1}^{n_i} x_{ij}^* \right\|. \tag{5.5}$$

The overall goal of this estimation procedure is to get a value for $\hat{\mu}_0$ as close as possible to the ground truth μ_0.

We assess the accuracy of the pole estimate across simulations by computing the root-mean-square error (RMSE) as

$$\text{Err}_{\hat{\mu}_0} = \sqrt{\frac{1}{M} \sum_{m=1}^{M} \text{angle} \left(\hat{\mu}_0^{(m)}, \mu_0 \right)^2}, \tag{5.6}$$

where $\text{angle}(\hat{\mu}_0^{(m)}, \mu_0) = (180°/\pi) \cos^{-1}(\hat{\mu}_0^T \hat{\mu}_0^{(m)})$ is the angular distance in degrees between the true pole μ_0 and each one of the simulated estimations $\hat{\mu}_0^{(m)}$, where M is the total number of simulations.

5.3.3 Estimation of the VGP scatter

Long-term assessment of the paleomagnetic secular variation of the geomagnetic field relies on the VGPs dispersion S_b instead of their mean. The observed global dispersion S is estimated as (Cox 1970)

$$\hat{S}^2 = \frac{1}{N-1} \sum_{i=1}^{N} \text{angle}(\hat{\mu}_i, \hat{\mu}_0)^2. \tag{5.7}$$

The global dispersion S^2 is a combination of the dispersion *between* VGPs S_b and that arising from the dispersion among the samples *within* the site S_w (McFadden et al. 1991). We assume that the latter arises purely from random errors associated with orientation, measurement and analytical errors, whereas the former is an unknown, latitude-dependent parameter of the time-averaged geomagnetic field. In order to estimate S_b, we first need to extract the within-site dispersion from the global dispersion of the VGPs, that is

$$\hat{S}_b^2 = \hat{S}^2 - \hat{S}_w^2, \tag{5.8}$$

where the estimated within-site dispersion $\hat{S}_w$ is computed in directional space following (McFadden et al. 1991) and (Doubrovine et al. 2019)

$$\hat{S}_w^2 = \frac{1}{N} \sum_{i=1}^{N} \frac{\hat{S}_{wi}^2}{n_i} \tag{5.9}$$

$$\hat{S}_{wi}^2 = 2 \left(\frac{180^\circ}{\pi} \right)^2 \frac{T(\lambda)}{\hat{k}_{wi}} \tag{5.10}$$

$$\hat{k}_{wi} = \frac{n_i - 1}{n_i - R_i}, \tag{5.11}$$

with $T(\lambda) = \frac{1}{8}(5 + 18 \sin^2 \lambda + 9 \sin^4 \lambda)$ the latitude correction introduced in (Cox 1970); and R_i the resultant vector length defined in Equation 5.5. Notice that the within-site dispersion will lead to unrealistic estimates of the between-site dispersion in cases where n_i is small, $n_i = 1$ being the extreme case where the within-site dispersion cannot be estimated; that is, we cannot disentangle the contribution of the within-site and between-site dispersion. For cases where $n_i = 1$, we set $\hat{S}_w = 0$, that is, the within site dispersion is zero since it cannot be estimated from these series of equations.

5.4 Numerical results

In this section, we present the results of numerical simulations that explore how different sampling strategies affect the estimation of paleopole position μ_0 and VGP scatter S_b. These simulations implement the data generation process described in the Section 5.3.1 to draw samples of site directions and associated directions within a given site. For the different numerical experiments, we apply varied choices for the model parameters (Table 5.1) and we respectively compute the mean pole position $\hat{\mu}_0$ and VGP scatter $\hat{S}_b$. These simulations enable us to assess what differences in sampling strategy yield estimates of the parameters of interest that are closer to the true value. We compare the results of these estimates for different choices of filters to determine which sampling strategy and method yields the highest accuracy.

5.4.1 Trade-off between number of sites and number of samples per site

The top panel in Figure 5.2 shows the accuracy of the mean $\hat{\mu}_0$ (Equation (5.6)) as a function of the number of sites N and the number of samples per site n_0 in the absence of outliers ($p_{\text{outlier}} = 0$). As the number of sites increases (moving up the y-axis), the total error reduces. The mean error is also reduced if we increase the number of samples per site while keeping the total number of sites fixed. However, in the latter case we see that the improvement resulting from increasing the number of samples per site is small relative to increasing the number of sites and saturates for small numbers of n_0 (see black contour lines).

In a scenario with unlimited resources to collect and analyze paleomagnetic samples, one could seek to maximize both the number of sites (N) and the number of samples per site (n_0). However, in the context of finite resources, it is interesting to consider what happens when we keep fixed the total number of samples $n = n_0 N$ but change how these samples are partitioned between number of sites (N) and number of samples per site (n_0). As visualized with the white dotted curves in Figure 5.2 that follow a fixed total number of samples, we see that smaller errors are associated with sampling strategies that prioritize the acquisition of additional sites over the collection of additional samples per site. The same behaviour is exposed when we plot the error as a function of the total number of samples n and for different values of n_0 (Figures 5.3a and 5.3b). For all choices of samples per site n_0, the net error decreases at rate $1/\sqrt{n}$, with the absolute value of the error being additionally affected by n_0. We quantify the improvement in accuracy due to an increase in the number of samples for different number of samples per site (Figures 5.3c and 5.3d). Even by keeping fixed the number of sites and increasing n_0 (and, consequently, increasing the total number of samples), the improvement in accuracy is minimal once $n_0 \geq 3$.

The effect of varied numbers for N and n_0 on the accuracy of estimates of VGP scatter (between-site dispersion S_b) is shown in Figure 5.2. As with estimating pole position, we observe similar behavior for estimating VGP scatter where, given a fixed total number of samples, there is smaller error when the number of sites is higher. However, the benefit of increasing the number of samples per site on reducing the root mean square error between $\hat{S}_b$ and the true VGP scatter S_b is more pronounced. Notice that for $n_0 = 1$, this error is large due to the inability to estimate the within-site dispersion. However, for $n_0 \geq 3$ the error stabilizes and we observe the same behaviour as before: the acquisition of more sites over more samples per site leads to better estimation of the VGP scatter assuming $n_0 \geq 3$.

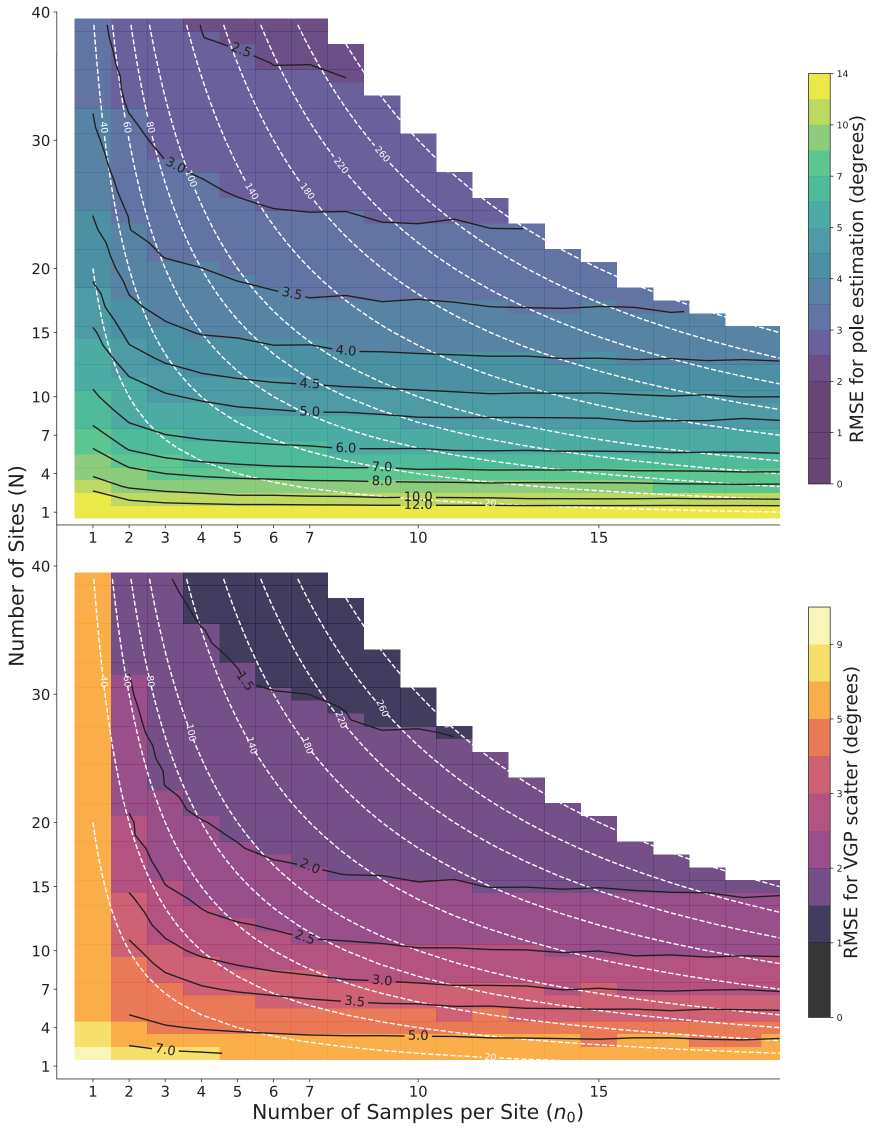
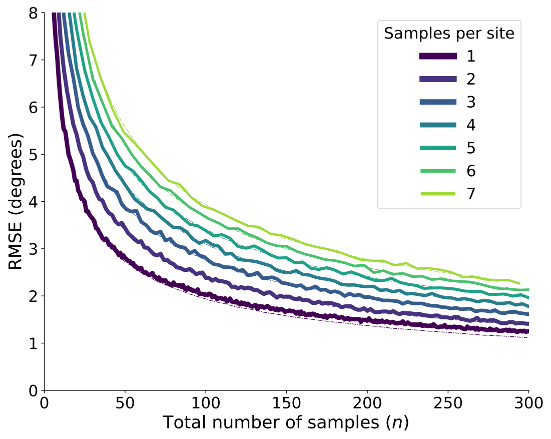
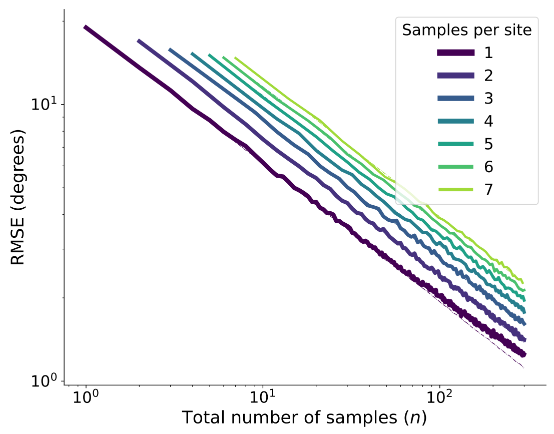


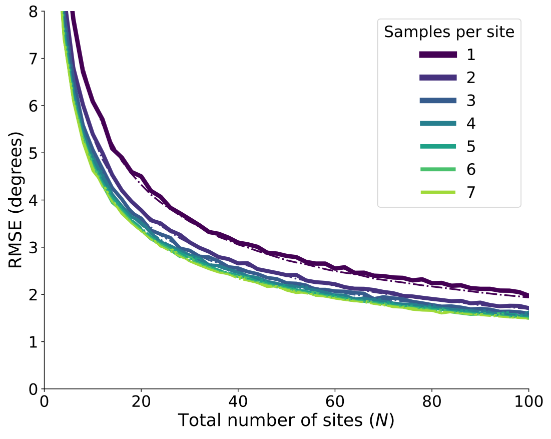
Figure 5.2: *Root mean square error (RMSE) in degrees between site mean poles and the true GAD pole (top panel) and between-site VGP dispersion (bottom panel) as a function of different combinations of the total number of sites N and the number of samples per site n_0. For this diagram, we use a paleolatitude of $30°$ ($\kappa_b \approx 35$), $p_{outlier} = 0$, and $\kappa_w = 50$. The white dashed lines represent isolines where the total number of samples n is constant, and the black lines represent isolines with constant net mean error angle. Each point-wise estimate of the mean error (i.e. each box) is based on the results of $10,000$ simulations. While these simulations represent secular variation using model G, similar results emerge from using the TK03 model (Tauxe et al. 2004).*



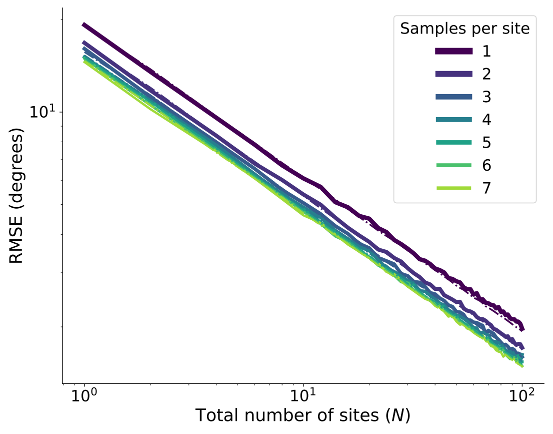
(a)



(b)



(c)



(d)

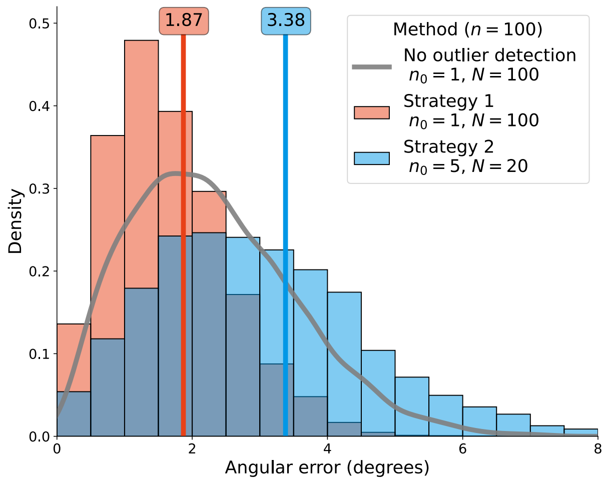
Figure 5.3: *(a) Root mean square error (RMSE) angle of the computed mean pole as a function of the total number of samples n for different values of samples per site n_0 where an increase in samples per site results in a decrease in the number of sites. (b) Displays the same values on a logarithmic scale, making explicit the $1/\sqrt{n}$ decay of the error, independent of the value of n_0. (c) RMSE as a function of the total number of sites N for different values of n_0 where an increase in n_0 increases the total number of samples, also in (d) logarithmic scale. For all the figures, we set $\lambda = 30^\circ$, $\kappa_w = 50$, and $p_{outlier} = 0$. The dot-dashed lines in all the plots represents the theoretical approximation (see Section 5.5).*

5.4.2 Sampling strategy in the presence of outliers

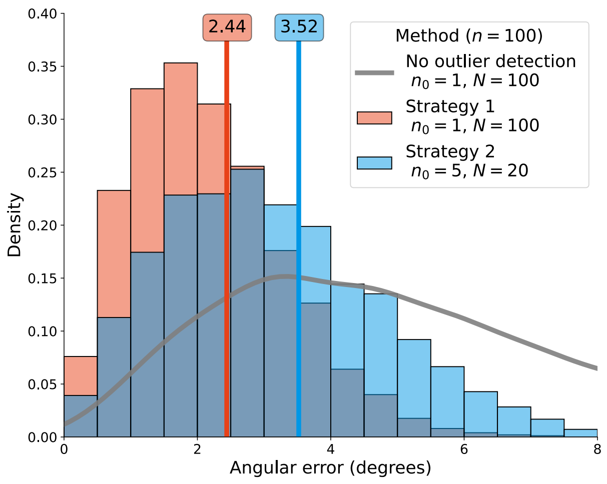
In the previous section, we concluded that the number of sites N is mostly what determines the accuracy of the estimated position of the paleopole. However, an argument for collecting more samples per site is the ability to detect and filter out spurious sample directions. A more fair comparison then is to compare two different strategies for estimating the paleopole while taking the possible occurrence of such outliers into account. When using a small number of samples per site n_0, outlier detection at the site level may be difficult, or directly impossible where $n_0 = 1$ given that within site consistency cannot be evaluated. However, it is possible to implement methods to filter VGPs that are statistically significantly apart from the mean (e.g. the paleopole) using an iterative cut-off (Vandamme 1994). We compare this first strategy ($n_0 = 1$ with Vandamme's iterative cut-off applied on the estimated population of VGPs) with the optimistic case where we collect more samples per site and are able to identify and filter all the outliers directly at the site level. The latter case provides a lower bound on the most optimistic error when using any outlier detection criteria at site level. For this second strategy, no outliers are included in the calculation of the final estimated pole $\hat{\mu}_0$. This means that the effective number of samples used to estimate μ_0 will be less than n, but since the samples removed are spurious directions, we expect the estimate of the paleopole will be more accurate than if we included all the samples in the calculation. We also show the results of the first method without using any outlier filter whatsoever.

Histograms in Figures 5.4 show the distribution of the angles between μ_0 (true GAD pole) and $\hat{\mu}_0$ (estimated pole) for the two sampling strategies and with 10%, 40% and 60% outlier rate, respectively. Even in the presence of outliers, using $n_0 = 1$ gives lower angular errors than when using $n_0 = 5$ until the proportion of outliers p_{outlier} increases by a significant amount. We illustrate this by showing in Figure 5.5a the mean of these two errors as a function of the outlier rate p_{outlier}. Until the proportion of outliers reaches a critical point of approximately 55%, having $n_0 = 1$ but being able to sample more sites N still out-performs the case where $n_0 = 5$ and all outliers are removed. Figure 5.5b shows this critical value of p_{outlier} for different site latitudes and within-site dispersion, showing that we need to have more than 40% outliers before the second strategy out-performs the $n_0 = 1$ strategy. Panel 5.5c further shows this critical value in the case where no filter is used for $n_0 = 1$. It is noteworthy that despite the small variance, this critical value of p_{outlier} grows as a function of site latitude (increasing S_b) and remains relatively similar as a function of within-site dispersion.

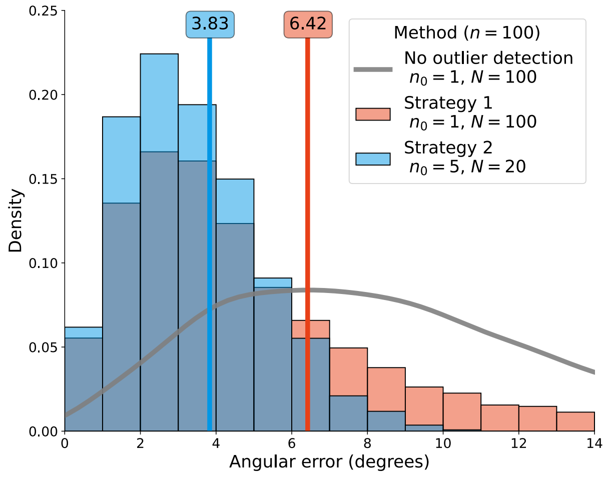
A wider comparison of these methods for a range of samples per site n_0 is provided in Figure 5.6. Here again we can observe that for a fixed number of total samples the scenario with $n_0 = 1$ leads to better estimation of the true pole until the proportion of outliers becomes very high. On the right side of the panel we can also observe the improvement in accuracy when we fix the number of sites N and we increase the number of samples per site and thus the total number of samples. In agreement with the results shown in Figure 5.3, we observe that the improvement due to an increase in the number of samples per site n_0 by keeping N fixed is small compared to a change in the overall sampling strategy.



(a) 10% Outliers

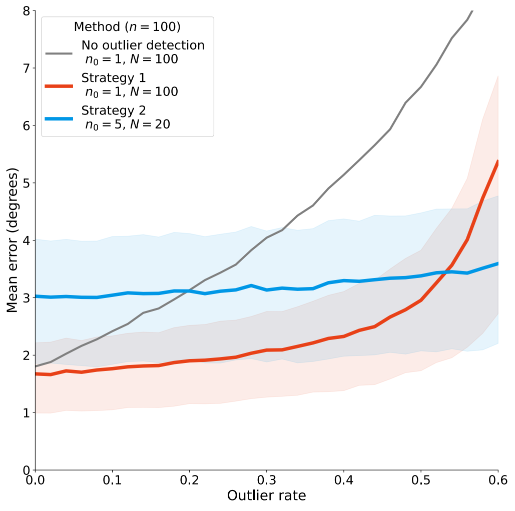


(b) 40% Outliers

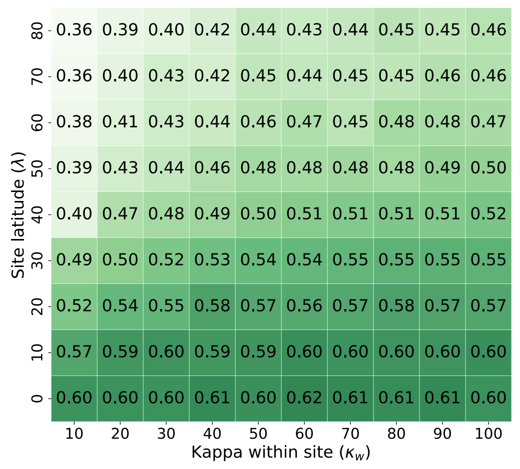


(c) 60% Outliers

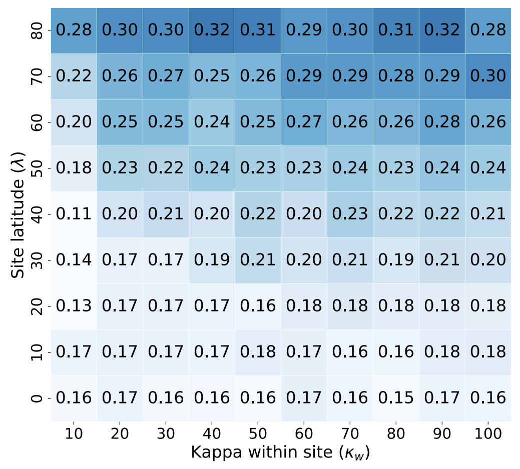
Figure 5.4: *Comparison between two different sampling strategies to determine a mean paleomagnetic pole position in the presence of outliers for a fixed number of total samples ($n = 100$). The red histograms and curve are strategy 1 where we have one sample per site ($n_0 = 1$), one hundred sites ($N = 100$) and we use the Vandamme filter. The blue histograms and curve are strategy 2 where $n_0 = 5$, ($N = 20$) and we filter all the outliers (perfect detection algorithm) for (a) $p_{outlier} = 0.10$ (10% of sample directions are outliers); (b) $p_{outlier} = 0.40$; and (c) $p_{outlier} = 0.60$. Here $\kappa_w = 66$ is such that the angular dispersion within site is $10°$, and $\lambda = 30°$. The gray line denotes the case in which we sample for $n_0 = 1$ but we do not use any outlier detection method.*



(a) Intersecting errors



(b) p_{outlier} critical



(c) p_{outlier} critical

Figure 5.5: *Comparison between two different sampling strategies to determine a mean paleomagnetic pole position in the presence of outliers for a fixed number of total samples ($n = 100$) (Part 2). (a) As we increase the number of outliers $p_{outlier}$, the error increases differently depending on whether we can detect and filter outliers or not. The intersection of the two errors corresponds to the value of $p_{outlier}$ whereupon there is a crossover in the efficacy of the two methods. The shaded envelopes around the solid lines correspond to the 25 and 75 percentile bands. (b) Value of the intersection between the mean errors for strategies 1 and 2 (panel a) for different values of latitude λ and within-site dispersion k_w. (c) Same as in (b) but comparing $n_0 = 5$ with the scenario of no outlier detection.*

We conducted the same analysis for estimating the VGP scatter S_b and its associated error. Figure 5.7 shows the signed percentage error $100\% \cdot (\hat{S}_b - S_b)/S_b$ for different choices of n_0. When $n_0 = 1$, all methods overestimate the real VGP scatter due to the lack of estimates of the within site dispersion S_w^2 (Equation (5.9)). On the other hand, S_b tends to be underestimated when we use the (Vandamme 1994) filter, since the cut-off of outliers reduces the total dispersion of the VGPs (Equation (5.7)). As we increase the number of outliers, we observe a significant deterioration of the VGP scatter estimation due to the inability to filter outliers. This behaviour is rather different to what we observed for paleopole estimation, where the estimation is more robust to outliers. However, after reaching a minimum required value of samples per site (around $n_0 = 3$), the accuracy only minimally improves by adding more samples per site. In the case where no outliers are present, we are back to the case in Figure 5.2 where we observed that, for the same budget of total samples n, a larger value of sites N leads to more accurate estimates as long as $n_0 \geq 3$.

5.5 Theoretical results

We can quantify the trade-offs between the different model parameters introduced in the previous section by theoretically deriving approximations for the dispersion parameter of the distribution of the estimated pole $\hat{\mu}_0$. This procedure works by finding the effective precision parameter κ_{eff} of a Fisher distribution that minimizes the Kullback-Leibler divergence with respect to the actual dispersion of $\hat{\mu}_0$ (Heslop et al. 2020; Kurz et al. 2016). As derived in (Kurz et al. 2016), this approach is equivalent to finding the mean direction and dispersion parameter that matches the resultant vector length of the target distribution. Using this method, we will derive in this section the following approximation for the dispersion of the estimated $\hat{\mu}_0$:

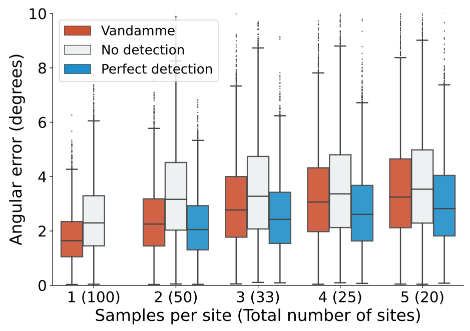
$$\hat{\mu}_0 \approx \text{Fisher}(\mu_0, \kappa_{\text{eff}}), \qquad \kappa_{\text{eff}} = \frac{N\kappa_b}{1 + \frac{\kappa_b}{n_0\,(1-p_{\text{outlier}})\,\kappa_w\,T(\lambda)}}. \tag{5.12}$$

The effective dispersion parameter κ_{eff} is a function of all the parameters in the model. Under the assumptions of model G (McFadden et al. 1988), we have $\kappa_b = \kappa_b(\lambda)$ is a function of the paleolatitude according to Equation (5.2). However, this result holds for other choices of κ_b where the Fisher approximation of the VGP scatter is appropriate.

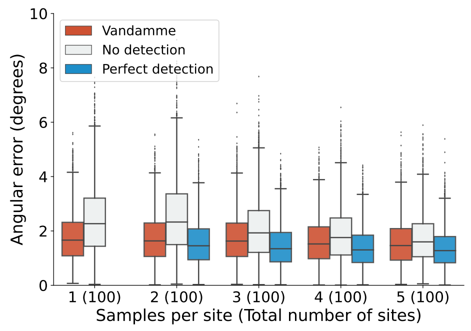
In the case where no outliers are included ($p_{\text{outlier}} = 0$), based on the approximated relationship between angular dispersion S and κ we can approximate the angular error $\text{Err}_{\hat{\mu}_0}$ introduced in Equation (5.6) as

$$\text{Err}_{\hat{\mu}_0} \approx \frac{81^\circ}{\sqrt{N}} \sqrt{\frac{1}{\kappa_b} + \frac{1}{n_0 \kappa_1 T(\lambda)}}. \tag{5.13}$$

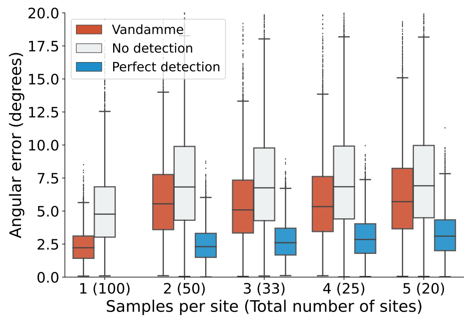
This equation allow us to quantify the amount of error associated with different choices of n_0. Comparing this theoretical approximation with the simulations (Figure 5.2 and 5.3) reveals relative error of around 1% between simulation and theory.



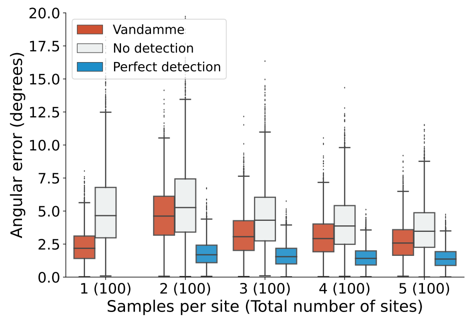
(a) $n \approx 100$, 10% Outliers



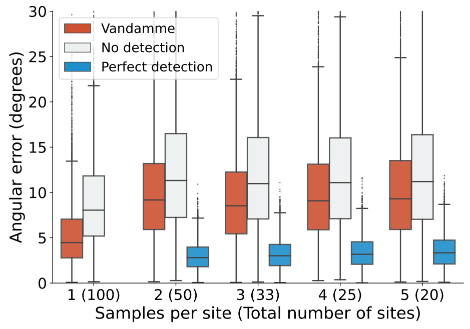
(b) $N = 100$, 10% Outliers



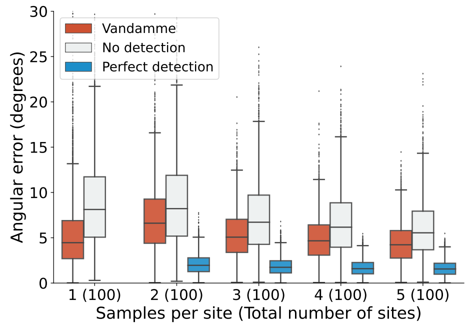
(c) $n \approx 100$, 40% Outliers



(d) $N = 100$, 40% Outliers

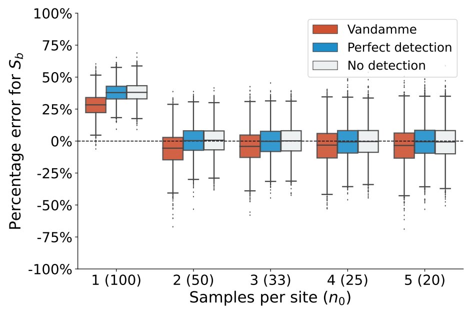


(e) $n \approx 100$, 60% Outliers

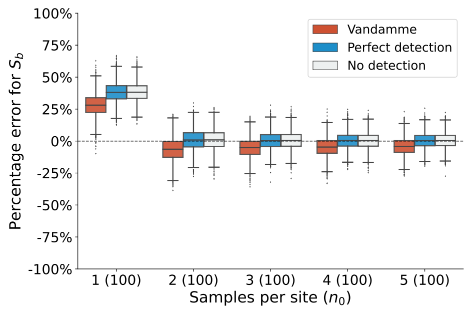


(f) $N = 100$, 60% Outliers

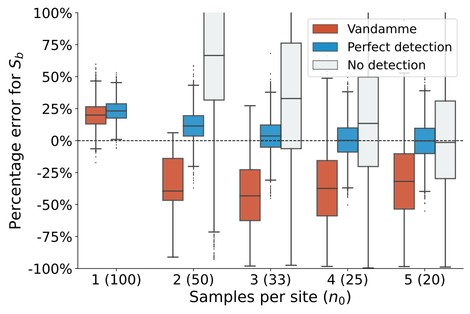
Figure 5.6: *Boxplot of the angular error between estimated and true GAD pole for different sampling strategies (number of samples per site, and total number of sites in parenthesis) for (a,b) $p_{outlier} = 0.10$, (c,d) $p_{outlier} = 0.40$ and (e,f) $p_{outlier} = 0.60$. The left column corresponds to the case where the total number of samples is fixed around $n \approx 100$, while the right column is the case with fixed number of sites ($N = 100$) and a variable total number of samples. Following the convention in Figure 5.4, the red diagrams correspond to $n_0 = 1$ using the Vandamme filter; the blue to $n_0 = 5$ with perfect outlier detection algorithm; and the grey boxes correspond to $n_0 = 1$ with no outlier detection been applied. For all simulations shown, $k_w = 50$ and $\lambda = 30°$.*



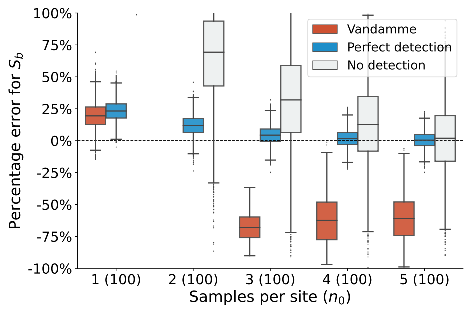
(a) $n \approx 100$, 0% Outliers



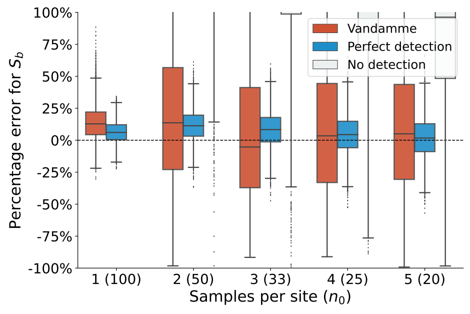
(b) $N = 100$, 0% Outliers



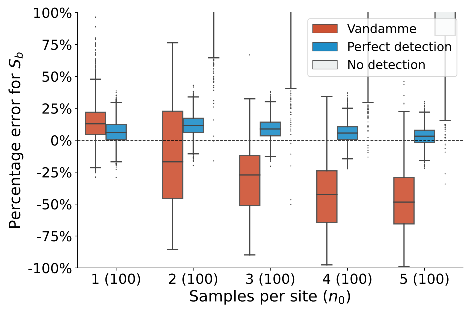
(c) $n \approx 100$, 20% Outliers



(d) $N = 100$, 20% Outliers



(e) $n \approx 100$, 40% Outliers



(f) $N = 100$, 40% Outliers

Figure 5.7: *Boxplot of the relative error when estimating the between-site dispersion S_b, that is, $100\%(\hat{S}_b - S_b)/S_b$, where $\hat{S}_b$ is estimated as it was explained in Section 5.3.3, and S_b is the true VGP scatter. Parameters, color references and panel arrangements are the same than in Figure 5.6, while here the choice of outliers rates is (a,b) $p_{outlier} = 0$, (c,d) $p_{outlier} = 0.20$ and (e,f) $p_{outlier} = 0.40$.*

From the theoretical expression for $\text{Err}_{\hat{\mu}_0}$ we can see that as n_0 increases, the improvement in accuracy to the final error becomes rather minimal since the coefficient $1/n_0\kappa_1 T(\lambda)$ is dominated by $1/\kappa_b$. Surprisingly, this limit is reached for very small values of n_0, which shows the small amount of improvement that increasing n_0 adds to the final error, especially when we compare this with the decay of the error given by the factor $1/\sqrt{N}$. No matter the choice of n_0, the error goes to zero as N increases. On the other hand, no matter how large n_0 becomes, the overall error will never be lower than $81^\circ/\sqrt{N\kappa_b}$, N being the quantity that controls the overall error most.

The approximation with outliers is accurate for values of which $n_0(1-p_{\text{outlier}})$ is strictly larger than one. For the case of $n_0 = 1$, a more accurate approximation is given by

$$\rho^{-1}\left((1-p_{\text{outlier}}) \frac{N\kappa_b}{1+\frac{\kappa_b}{n_0\,\kappa_w\,T(\lambda)}} \right), \tag{5.14}$$

where $\rho(\kappa) = 1/\tanh(\kappa) - 1/\kappa$ is the expected length of a Fisher distribution with precision parameter κ and ρ^{-1} its inverse. When using a perfect outlier algorithm with $(1-p_{\text{outlier}})n_0 \geq 2$, the approximation in Equation (5.13) is still appropriate. Further investigation is needed to estimate the final error when using iterative cut-off methods such as the Vandamme filter (Vandamme 1994).

Notice that the theoretical expression for the final dispersion can be used to define confidence intervals around the true pole for a specific study case. Effectively, given a sampling procedure with prescribed N and n_0, we can estimate the dispersion parameters κ_w and κ_b and then, by plugging these into Equations (5.12) and (5.13) obtain a confidence region around the sample estimated pole. This procedure will take into account the hierarchical nature of paleomagnetic samples at the moment of quantifying uncertainty.

5.5.1 Setup

The building blocks that lead to that final results in Equations (5.12), (5.13), and (5.14) consist in finding approximate Fisher distributions for the following procedures:

1. Mean of Fisher distributions (Section 5.5.2)
2. Hierarchical sample of two nested Fisher distributions (Section 5.5.3)
3. Superposition of Fisher and uniform distributions (Section 5.5.4).

Just as we assumed before, we randomly sample a total of N VGPs μ_i in latitude-longitude space from a Fisher distribution with mean μ_0 and concentration parameter κ_b. Then, we sample site measurements x_{ij}^* in directional space from a Fisher distribution with mean μ_i^* and concentration parameter κ_w, where $j = 1, 2, \ldots, n_i$ (see Section 2.1). We are going to use $\rho(\cdot)$ to refer to the function

$$\rho(\kappa) = \frac{1}{\tanh(\kappa)} - \frac{1}{\kappa}, \tag{5.15}$$

where κ will refer to the precision parameter of Fisher distributions. It is easy to see that $\rho(\kappa)$ is the expected length of a draw from a Fisher distribution with concentration parameter κ (Mardia et al. 2000).

The method for approximating Fisher distributions follows the moment matching procedure also used in (Heslop et al. 2020). If $p(x)$ represents the probability density function of some random estimate with support in the unit-sphere given by $\mathcal{S}^2 = \{x \in \mathbb{R}^3 : \|x\| = 1\}$, then we aim to find the parameters $\mu \in \mathbb{R}^3$ ($\|\mu\| = 1$) and κ of the Fisher probability density function $q(x; \mu, \kappa)$,

$$q(x; \mu, \kappa) = \frac{\kappa}{4\pi \sinh(\kappa)} e^{\kappa \mu^T x}, \tag{5.16}$$

such that they minimize the Kullback–Leibler divergence $D_{KL}(p|q)$ given by

$$\min_{\mu,\kappa} D_{KL}(p|q) = \int_{\mathcal{S}^2} p(x) \log \frac{p(x)}{q(x; \mu, \kappa)} dx. \tag{5.17}$$

As it was found in (Kurz et al. 2016), this is equivalent to finding a Fisher distribution $q(x; \mu, \kappa)$ with same mean direction and mean vector length, where the mean vector (both direction and length) is computed as $\int_{\mathcal{S}^2} x\, p(x)\, dx$. The technique then consists in estimating the mean resultant length of the estimated paleopole $\hat{\mu}_0$ and matching it with the corresponding Fisher distribution $q(x; \mu, \kappa)$ with same mean resultant length.

5.5.2 Mean of Fisher distributions

Let us begin with a result about the distribution of the mean of a total of n Fisher distributions with same dispersion parameter κ. The case $n = 1$ is excluded since it leads to a trivial result.

Proposition 1 (Mean of Fisher Distributions). *Consider a sample of $n \geq 2$ independent Fisher distributions x_i, $i = 1, 2, \ldots, n$, with mean μ_0 and precision parameter κ. Then the Fisher mean*

$$\hat{\mu} = \frac{1}{nR} \sum_{i=1}^{n} x_i \qquad R = \left\| \frac{1}{n} \sum_{i=1}^{n} x_i \right\| \tag{5.18}$$

is approximately Fisher distributed with mean direction μ_0 and precision parameter $\kappa n \rho(\kappa)$.

Proof. Following (Fisher 1953), the estimated mean Fisher distribution $\hat{\mu}$ can be approximated with a Fisher distribution with mean μ and concentration parameter $\kappa n R$. This can be derived by the fact that the conditional probability of $\hat{\mu}|R = r$ has distribution Fisher$(\mu_0, n\kappa r)$ (Mardia 1975). Taking then expectation over R we obtain an approximate value of the effective concentration parameter of $\hat{\mu}$. Now, we need to find the expected value of the vector length of the mean estimate R. For $n > 1$, it is easy to see that this last quantity coincides in expectation with the expected length of the Fisher distribution with parameter κ, that is $\mathbb{E}[R] = \rho(\kappa)$ (Heslop et al. 2020; Mardia et al. 2000). On the other side, for $n = 1$ we simply have $R = 1$. Even when this may be seem clear, let us derive a secondary proof that will be useful in the later section. For distributions in the sphere, the following relationship holds ((Mardia et al. 2000), equation 9.2.13)

$$\mathbb{E}\left[R^2\right] = \mathbb{E}\left[R\right]^2 + \frac{1}{n}(1 - \mathbb{E}\left[R\right]^2), \tag{5.19}$$

or equivalently

$$\mathbb{E}\left[R\right]^2 = \frac{n\,\mathbb{E}\left[R^2\right] - 1}{n-1}. \tag{5.20}$$

This last equation is useful because it allow us to compute the expected value of R as a function of the expected value of R^2, which is mathematically easier to manipulate. Now,

$$R^2 = \frac{1}{n^2}\sum_{i,j=1}^{n} x_i^T x_j = \frac{1}{n^2}\sum_{i=1}^{n} \|x_i\|^2 + \frac{1}{n^2}\sum_{i\neq j} x_i^T x_j. \tag{5.21}$$

Now, taking expectation and using $\|x_i\| = 1$ and the independence of the x_i we have

$$\mathbb{E}\left[R^2\right] = \frac{1}{n} + \frac{n-1}{n}\,\mathbb{E}\left[x_1^T x_2\right]. \tag{5.22}$$

The only thing that remains to be calculated is the expectation of the cosine of the angle between independent Fisher distributed vectors $x_1^T x_2$. However, notice

$$\mathbb{E}\left[x_1^T x_2\right] = \rho(\kappa)\,\mathbb{E}\left[\mu_0^T x_2\right] = \rho(\kappa)^2 \tag{5.23}$$

which then leads to

$$\mathbb{E}\left[R^2\right] = \frac{1}{n} + \frac{n-1}{n}\rho(\kappa)^2 \tag{5.24}$$

and $\mathbb{E}[R] = \rho(\kappa)$. Finally, we have that $\hat{\mu}$ is approximately Fisher distributed with mean μ and expected concentration parameter equal to $\kappa n \rho(\kappa)$. □

5.5.3 Hierarchical sampling of Fisher distributions

Now, let us consider the case where we hierarchically sample Fisher distribution with random mean directions. This emulates the hierarchical computation of mean directions used to estimate paleopole directions.

Proposition 2 (Hierarchical Sampling on Fisher Distributions)**.** *Consider the following hierarchical sampling of Fisher distributed random variables.*

$$\begin{aligned}\mu_1 &\sim \textit{Fisher}(\mu_0, \kappa_0)\\ x &\sim \textit{Fisher}(\mu_1, \kappa_1)\end{aligned} \tag{5.25}$$

Then the full distribution of x can be approximated by a Fisher distribution with mean μ_0 and precision parameter κ^* equal to

$$\kappa^* = \frac{\kappa_0\kappa_1}{\kappa_0+\kappa_1}. \tag{5.26}$$

Proof. We can write the full probability density function of x by integrating the product of conditional densities over all the possible values of μ_1 in the sphere, that is

$$\begin{aligned} p(x) &= \int_{\mathcal{S}^2} p(x|\mu_1)p(\mu_1|\mu_0)d\mu_1 \\ &= \frac{\kappa_0\kappa_1}{(4\pi)^2\sinh(\kappa_0)\sinh(\kappa_1)}\int \exp\left\{\kappa_0\mu_0^T\mu_1+\kappa_1x^T\mu_1\right\}d\mu_1 \\ &= \frac{\kappa_0\kappa_1}{4\pi\sinh(\kappa_0)\sinh(\kappa_1)}\frac{\sinh(\|\kappa_0\mu_0+\kappa_1x\|)}{\|\kappa_0\mu_0+\kappa_1x\|}. \end{aligned} \tag{5.27}$$

Without lost of generality, we can assign $\mu_0 = (0,0,1)$ and then

$$\|\kappa_0\mu_0+\kappa_1x\| = \sqrt{\kappa_1^2x^2+\kappa_1y^2+(\kappa_1z+\kappa_0)^2} = \sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z}. \tag{5.28}$$

Now, we need to find the first moment of the previous distribution in order to compute the mean length, which implies solving the integral

$$\int_{\mathcal{S}^2}\frac{\sinh(\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z})}{\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z}}zd\Omega = 2\pi\int_{-1}^{1}\frac{\sinh(\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z})}{\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z}}zdz. \tag{5.29}$$

Now, this last integral can be solved analytically as

$$\begin{aligned} &2\pi\int_{-1}^{1}\frac{\sinh(\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z})}{\sqrt{\kappa_1^2+\kappa_0^2+2\kappa_0\kappa_1z}}zdz \\ &= \frac{2\pi}{\kappa_0^2\kappa_1^2}\Bigg[(\kappa_0\kappa_1z+1)\cosh(\sqrt{\kappa_0+\kappa_1^2+2\kappa_0\kappa_1z}) \\ &- \sqrt{\kappa_0+\kappa_1^2+2\kappa_0\kappa_1z}\sinh(\sqrt{\kappa_0+\kappa_1^2+2\kappa_0\kappa_1z})\Bigg]_{z=-1}^{z=1} \\ &= \frac{2\pi}{\kappa_0^2\kappa_1^2}\Bigg((\kappa_0\kappa_1+1)\cosh(\kappa_0+\kappa_1)-(\kappa_0+\kappa_1)\sinh(\kappa_0+\kappa_1) \\ &- (\kappa_0\kappa_1-1)\cosh(|\kappa_1-\kappa_0|)+|\kappa_1-\kappa_0|\sinh(|\kappa_1-\kappa_0|)\Bigg), \end{aligned} \tag{5.30}$$

which lead to the fact that the expected length of the vector x is

$$\frac{\kappa_0\kappa_1}{2\kappa_0\kappa_1\sinh(\kappa_0)\sinh(\kappa_1)}\Big((\kappa_0\kappa_1+1)\cosh(\kappa_0+\kappa_1)-(\kappa_0+\kappa_1)\sinh(\kappa_0+\kappa_1)$$
$$-(\kappa_0\kappa_1-1)\cosh(|\kappa_1-\kappa_0|)+|\kappa_1-\kappa_0|\sinh(|\kappa_1-\kappa_0|)\Big)$$
$$\asymp 1-\frac{\kappa_0+\kappa_1}{\kappa_0\kappa_1}, \tag{5.31}$$

where we use $\sinh(\kappa)\asymp\cosh(\kappa)\asymp e^{\kappa}/2$ for κ large enough. Comparing with the equivalent vector length $\rho(\cdot)$, we obtain that the equivalent dispersion parameter κ^* for the superposition of two Fisher distribution is given by

$$\kappa^*=\frac{\kappa_0\kappa_1}{\kappa_0+\kappa_1}, \tag{5.32}$$

as we wanted to prove. □

Notice that under the approximation that the dispersion coefficient S can be approximated as

$$S^2\approx 2\left(\frac{180}{\pi}\right)^2\frac{1}{\kappa}, \tag{5.33}$$

we can then derive that the dispersion S_*^2 associated to κ^* can be approximated as

$$S_*^2\approx 2\left(\frac{180}{\pi}\right)^2\frac{1}{\kappa}=2\left(\frac{180}{\pi}\right)^2\frac{\kappa_1+\kappa_2}{\kappa_1\kappa_2}\approx S_1^2+S_2^2, \tag{5.34}$$

where S_1 and S_2 are the dispersion associated to Fisher distribution with precision parameters κ_1 and κ_2, respectively.

5.5.4 Ensemble of Fisher and uniform distributions

We will now consider how to approximate a superposition of Fisher and uniform distribution. This approximation is going to be much more limited than the other ones, due to the fact than a superposition of Fisher and uniform does not have a shape similar to a Fisher distribution. However, when many samples are consider and we are computing the mean of samples coming form this ensemble, this approximation is quite accurate.

Proposition 3 (Superposition of Fisher with Uniform distributions)**.** *Consider the model where we sample a total of n samples x_i, $i=1,2,\dots,n$, from a Fisher distribution with some probability $1-p_{outlier}$ and with uniform distribution with probability $p_{outlier}$:*

$$\begin{aligned} &x_i\sim \textit{Fisher}(\mu,\kappa)\quad \textit{with probability } 1-p_{outlier} \textit{ and}\\ &x_i\sim \textit{Unif}\quad \textit{otherwise, for } i=1,2,\dots,n,\end{aligned} \tag{5.35}$$

Then the Fisher mean $\hat{\mu}$ of the n samples can be approximated with a Fisher distribution with mean μ and precision parameter equal to $n(1-p_{outlier})\kappa\rho(\kappa)$.

Proof. In order to compute the dispersion parameter, we need to compute the approximated vector length resulting from adding together draws from the Fisher and uniform distribution. Given a total number of $n_0 \leq n$ points that are not outliers, a similar calculation as in the derivation of Equation (5.21) leads to

$$\mathbb{E}\left[R^2 | n_0\right] = \frac{1}{n^2}\left(n + n_0(n_0-1)\rho(\kappa)^2\right). \tag{5.36}$$

Now, using that n_0 has Binomial distribution with success probability $1 - p_{\text{outlier}}$ and a total of N samples, taking expectation over n_0 and noticing that $\mathbb{E}\left[n_0\right] = n(1-p_{\text{outlier}})$ and $\mathbb{E}\left[n_0^2\right] = np_{\text{outlier}}(1-p_{\text{outlier}}) + n^2(1-p_{\text{outlier}})^2$, we obtain

$$\mathbb{E}\left[R^2\right] = \frac{1}{n} + \frac{n-1}{n}(1-p_{\text{outlier}})^2\rho(\kappa)^2. \tag{5.37}$$

This leads to $\mathbb{E}\left[R\right] = (1-p_{\text{outlier}})\rho(\kappa)$ for $n \geq 2$. □

5.5.5 General Fisherian approximation of the pole mean

These last three results allow us to approximate a hierarchical sample of Fisher distributions with a very good level of accuracy. In order to compute the final dispersion of the pole, notice that each estimated VPG $\hat{\mu}_i^*$ in directional space can be approximated as a sample from a Fisher distribution with dispersion parameter

$$n_i\kappa_i(1-p_{\text{outlier}})\rho_{n_i}(\kappa_i), \tag{5.38}$$

where $\rho_n(\kappa) = \rho(\kappa)$ for $n \geq 2$ and $\rho_1(\kappa) = 1$ (Propositions 1 and 3). We have introduced this extra notation in order to include both the $n_i = 1$ and $n_i \geq 2$ cases in the same expression. Now, since the Fisher mean of the VGPs is computed in directional space, we need to include the latitude correction factor $T(\lambda)$ when we convert these to VGP space (Cox 1970). This then implies that we can approximate

$$\hat{\mu}_i \sim \text{Fisher}\Big(\mu_i, n_i\kappa_w(1-p_{\text{outlier}})\rho_{n_i}(\kappa_w)T(\lambda)\Big). \tag{5.39}$$

Finally, since μ_i (the mean direction for $\hat{\mu}_i$) is also Fisher distributed with mean μ_0 and precision parameter κ_b, using Proposition 2 we have that the final pole $\hat{\mu}_0$ will have dispersion parameter equal to

$$\frac{\kappa_b}{1 + \frac{\kappa_w}{\kappa_b(1-p_{\text{outlier}})n_i\rho_{n_i}(\kappa_w)T(\lambda)}} \tag{5.40}$$

Now, if $n_i = n_0$ are all the same, we can average all the $\hat{\mu}_i$ to came up with the final pole dispersion parameter

$$\hat{\mu}_0 \sim \text{Fisher}\left(\mu_0, \frac{N\kappa_b\rho_N(\kappa_b)}{1 + \frac{\kappa_b}{\kappa_w n_i(1-p_{\text{outlier}})\rho_{n_0}(\kappa_w)T(\lambda)}}\right). \tag{5.41}$$

Assuming $\rho(\kappa_i) \approx 1$, we then obtain that the final estimate $\hat{\mu}_0$ has a concentration parameter κ^* approximately equal to

$$\frac{N\kappa_b}{1 + \frac{\kappa_b}{\kappa_w(1-p_{\text{outlier}})n_0 T(\lambda)}}, \tag{5.42}$$

which is the same expression as in Equation (12). In order to derive Equation (13), we rely again in the approximation of the dispersion given in Equation (5.33).

As we mentioned before, Proposition 3 will fail when the number of samples per site n_0 is small and the number of outliers p_{outlier} is large. For those cases, a better approximation is given by Equation (14). This arises from computing the expected vector length without outliers and then multiply the expected vector length by the factor $(1 - p_{\text{outlier}})$, which gives an approximated vector length for this case. We then find the corresponding κ for such resultant length by computationally inverting the function $\rho(\kappa)$.

5.6 Recommendations

When the goal is to estimate the position of a paleopole, our results show that the total number of sites N has a far larger impact on accuracy than the number of samples per site n_0. We therefore recommend the following rule of thumb for sample collection where the objective is paleopole estimation: the more samples the better, but efforts to maximize the number of independent sites will have a greater effect on improving accuracy than more samples per site. In particular, the benefit of collecting more samples per site is small for $n_0 \geq 3$ and diminishes at $n_0 \geq 5$. Analyzing more samples than these values per site is inadvisable if it will result in fewer overall sites in a given study. As was concluded in (Gerritsen et al. 2022), for the purpose of computing a paleopole and for a fixed total number of samples, it is always better to collect these samples from different sites than to collect more samples per sites. In the context of sedimentary sections, this result strongly supports stratigraphic sampling strategies of one sample per horizon for directional estimation where each sample is its own site, consistent with previous findings by (Vaes et al. 2021). Collecting a large number of single-sample sites is also beneficial for the application of the elongation-inclination (E/I) correction for inclination shallowing (Tauxe et al. 2004), which requires $N \geq 100$ to be robust. In settings of limited sites or where moving between sites is itself resource intensive, as can be the case of igneous intrusions, there is a benefit to more samples per site given that it can improve site level direction estimates and enable within site outlier detection.

A recent approach to synthesize site data into apparent polar wander paths developed by (Gallo et al. 2023) enabled the propagation of directional uncertainty using site level precision κ_w estimated by multiple samples in a given site. This approach is not possible when applying a $n_0 = 1$ sampling strategy. However, estimation of the in-site dispersion can be derived using a different estimator such as the maximum angular deviation (MAD) of a directional fit (Khokhlov et al. 2016).

For paleopole estimates, filters based on populations of VGPs can aid in the detection of outliers (Vandamme 1994). If there is an appreciable outlier rate, such filtering schemes are necessary when $n_0 = 1$ given that outliers cannot be detected through within site consistency. When conducting a study with a low number of samples per site, the site consistency test proposed by (Gerritsen et al. 2022) can be applied where more samples are analyzed for selected sites. This field test can be used to gain insight into within site reproducibility and precision for a given lithology. Multiple samples per site can also be advisable when the presence of single outliers would have a major impact on interpretations such as in the case of interpreting geomagnetic polarity or transitional directions. We recommend that researchers use of Equation (5.13) to obtain an estimate of the net error as a function of the expected parameters present in the sampling.

An important caveat concerning the use of directional filters is that while the mean may be relatively insensitive to their effects, they can significantly distort the shape of the true directional distribution and should therefore be avoided where the latter is a parameter of interest (e.g. paleosecular variation studies). Indeed, the presence of outliers has a major impact on the estimation of the dispersion, and thus the VGP scatter S_b. Increasing the number of samples per site n_0 is beneficial as long as this helps us to detect outliers more accurately. However, this is not always straightforward using conventional data filters and cutoffs, which leads to a reliance on the expert's subjective interpretation (Gerritsen et al. 2022). There is a greater improvement in the accuracy of estimates of VGP scatter through increasing the number of samples per site, even in the absence of outliers, than there is for estimating the mean pole position. However, the improvement in the estimate of the VGP scatter progressively diminishes for increasing samples per site. When outliers can be detected efficiently, and for a minimum of three or four samples per site, the same trade-offs as noted above for paleopole estimation again apply: the preferential collection of more sites over more samples per site leads to more accurate estimates of the VGP scatter. And again, the most optimal sampling scheme given any suite of expected parameters can be determined from the results presented herein.

For general calculations of pole and VGP scatter accuracy, we recommend the interested reader to run their own experiments directly from the source code, which can be executed directly from the cloud using the provided Binder link (Jupyter et al. 2018) in the Code Availability section (Sapienza et al. 2023b).

5.7 Conclusions and future directions

The hierarchical nature of sampling in paleomagnetic investigations is a long-standing practice, but the community's specific default sampling strategies have largely relied upon conventional wisdom. Here we quantitatively explored, both numerically and analytically, the impact of different sampling strategies on the accuracy of estimates of paleopole position and VGP scatter. Our results demonstrate that when the objective is to estimate the position of the time-averaged paleomagnetic pole, a strategy that maximizes the number of sites is always the most favorable. Thus, given an infinite number of possible sites, it would be advantageous to collect as many single-sample sites as possible such as sampling one sample per stratigraphic horizon.

Where an estimate of VGP scatter is sought, the situation changes and the collection of single-sample sites hinders the estimation and exclusion of within-site directional scatter. The use of directional filters such as that of (Vandamme 1994) can lead to large inaccuracies in estimates of paleosecular variation. Here the optimal sampling strategy is more nuanced and the ideal number of samples per site depends on the expected proportion of outliers. However, the same general rule of thumb still applies: beyond some minimum number of samples per site the collection of additional sites should be prioritized over the collection of additional within-site samples.

We also emphasize that beyond these general rules of thumb, we herein provided tools enabling quantitative sampling recommendations to be generated from user-provided expectations. While specific project goals and geologic complexity should factor into project design we hope that these findings may free the community from the adoption of default sampling practices, and utilize statistically-informed strategies.

5.7.1 Universal differential equations in paleomagnetism

Finding smooth approximations to points distributed in the unit sphere plays an important role in directional statistics. An application is the estimation of apparent polar wander paths (APWP) in paleomagnetism used to describe the relative continental drift of tectonic plates. APWPs are extensively used to reconstruct past location of continents and are the only available method for performing paleography before 300 millions years ago (which accounts for 93% of Earth's history!).

As we discussed in Chapter 1 and as shown in 4, there has been an increasing interest in recent years in combining machine learning algorithms with differential equations. Solutions of differential equations can be used to represent a broad class of functions that can be used for regression purposes. This is the case with universal differential equations (UDEs) and neural differential equations, where observations are modelled as the numerical solution of a differential equation that has embedded some form of regressor (for example, a neural network) inside the differential equation (Rackauckas et al. 2020).

An application of this approach in paleomagnetism is currently being implemented in `SphereUDE.jl` (Sapienza et al. 2024b). Our model addresses the following three major points for data supported in a sphere. We further believe these considerations are a first step towards more complex models that combine statistical regression with differential equations.

1. **Sphere-constrained regression.** In this work we are interested in modelling data supported in the unit sphere.

2. **Path regularization.** We will start considering the general case on non-parametric regression with no regularization, but we are going to consider different types of regularization that are desired at the moment of fitting APWPs.

3. **Temporal uncertainties.** Ideally, we want a model that can deal with uncertainties in both space and time. Here we want to emphasize the importance of accounting for uncertainties in time, since paleopole dating is a difficult task and time estimates tend to carry important temporal uncertainties.

Consider a sequence of pairs (t_i, y_i), $i = 1, 2, \dots, N$, where each t_i corresponds to an observed time and each $y_i \in S^2$ is a unit vector supported in the unit sphere $S^2 = \{x \in \mathbb{R}^3 : \|x\|_2 = 1\}$. Without loss of generality, we assume $\mathrm{t}_{\min} \leq t_1 \leq t_2 \leq \dots \leq t_N \leq \mathrm{t}_{\max}$ for some $\mathrm{t}_{\min}, \mathrm{t}_{\max} \in \mathbb{R}$. A rotation matrix $R(p, \theta) \in \mathbb{R}^{3\times 3}$ is defined by an axis of rotation $p \in S^2$, called Euler pole, and an angle of rotation θ. Given some vector x_0, the rotation matrix $R(p, \theta)$ transforms x_0 into $R(p, \theta)x_0$. The space of rotations in three dimensions $SO(3)$ is a Lie group with an associated Lie algebra associated to the infinitesimal generators of rotations. A rotation matrix can be approximated at first-order with respect to the rotation angle as

$$R(p, \theta) = I + \theta \operatorname{skewt}(d) + \mathcal{O}(\theta^2), \tag{5.43}$$

where $\operatorname{skewt}(p)$ represents the skew-symmetric matrix defined by the rotation axis p, which satisfies $\operatorname{skewt}(p)x = p \times x$ for all vector $x \in \mathbb{R}^3$.

We will assume that each y_i is a sample from the von Mises-Fisher distribution (Fisher 1953; Watson 1982) in the sphere with mean direction $x(t_i)$ and concentration parameter κ_i, which controls the concentration of the distribution around the mean direction,

$$p(y_i | x(t_i), \kappa_i) = \frac{\kappa_i}{4\pi \sinh(\kappa_i)} e^{-\kappa_i x(t_i)^T y_i}, \tag{5.44}$$

where $x(t)$ is the solution of a differential equation.

There is a rich literature in regression problems where the goal is to learn trajectories in the sphere (Buss et al. 2001; Fletcher 2013; Hüper et al. 2007; Jupp et al. 1987; Marzio et al. 2019; Rosenthal et al. 2014; Samir et al. 2012). The novelty of our approach relies in the mathematical and computational simplicity of the method and its flexibility to incorporate new model requirements.

Sphere-constrained regression. Equation (5.43) represents the functional form of an infinitesimal rotation in the sphere. If we call $\theta = \omega \Delta t$, with ω being some angular velocity and Δt some time interval over which we apply the rotation, as $\Delta t \to 0$, the successive application of a rotation matrix with the same Euler pole p corresponds to solutions of the differential equation

$$\frac{dx}{dt}(t) = L(t) \times x(t) \tag{5.45}$$

where $L(t) = \omega(t)p(t)$ can be associated to the angular momentum of the solid sphere at some given time t, that is, a vector aligned with the rotation axis and with norm equals to ω. In fact, every smooth path in the sphere can be represented as the solution of such differential equation for some function $L : [\mathrm{t}_{\min}, \mathrm{t}_{\max}] \mapsto \mathbb{R}^3$, a result that is encapsulated in the following lemma.

Lemma 4. *For every continuous curve $y : [\mathrm{t}_{min}, \mathrm{t}_{max}] \mapsto S^2$ in the sphere, there is a function $L : [\mathrm{t}_{min}, \mathrm{t}_{max}] \mapsto \mathbb{R}^3$ such that $y(t)$ is the solution to the ordinary differential equation*

$$\frac{dx}{dt}(t) = L(t) \times x(t) \qquad x(\mathrm{t}_{min}) = y(\mathrm{t}_{min}) \tag{5.46}$$

Proof. Consider $L(t) = \alpha(t)x(t) + x(t) \times \dot{x}(t)$ for any function $\alpha : [\mathrm{t}_{\min}, \mathrm{t}_{\max}] \mapsto \mathbb{R}$ function. Using basic properties of the cross product, we have

$$L(t) \times x(t) = (x(t) \times \dot{x}(t)) \times x(t) = \dot{x}(t)\|x(t)\|_2^2 - x(t)(x(t) \cdot \dot{x}(t)). \tag{5.47}$$

Since $x(t) \in S^2$, we have $\|x(t)\|_2 = 1$ and $x(t) \cdot \dot{x}(t) = 0$. Notice that any value of $\alpha(t)$ will satisfy the differential equation , meaning that there are many $L(t)$ that will satisfy the differential equation. This is because infinitesimal variations of a path in a sphere are isomorphic to the tangent plane of S^2 at $x(t)$, instead of all the full group $\mathfrak{so}(3)$. However, notice that $\alpha \equiv 0$ is the one that makes $L(t)$ to have minimal Euclidean norm. □

This previous result give us a general formula to parametrize curves in the sphere. Given observations $y_1, y_2, \dots, y_N$, we can perform non-parametric maximum likelihood estimation by solving

$$\max_{L:\mathbb{R}\mapsto\mathbb{R}^3, x_0} \prod_{i=1}^{N} p(y_i | x(t_i), \kappa_i), \tag{5.48}$$

where $x(t)$ is the solution to the differential equation (5.47). This optimization problem is equivalent to *trajectory matching* (Ramsay et al. 2017)

$$\min_{L(t), x_0} \sum_{i=1}^{N} -\kappa_i \, y_i^T \, \mathrm{ODESolve}(t_i; x_0, L(t)), \tag{5.49}$$

where $\mathrm{ODESolve}(t_i; x_0, L(t))$ is the numerical solution of the differential equation (5.45) with initial condition $x(t_0) = x_0$ and time-dependent angular momentum $L(t)$.

In order to solve this optimization problem, we are going to parametrize the function $L : [t_0, t_N] \mapsto \mathbb{R}^3$ with a small neural network. Any other regressor with good expressivity that satisfies that the output is differentiable with respect to its arguments and parameters would serve as well. We will then write $L(t) = L(t; \theta)$ to parametrize the space of all possible time-varying angular momentum as a function of some high-dimensional parameter θ (for the case of a neural network, this will correspond to weights and biases).

Path regularization. Depending on the structure we want to impose to the path, we may be interested in imposing different types of regularization or constraints in the curves we parametrize in the sphere. This can be driven by domain knowledge of how the path should look (something we will further explore with the paleomagnetic data) or in order to break the multiplicity of solutions $L(t)$ that give rise to the same path $x(t)$. Expressing curves in S^2 as solutions of the differential equations (5.45) where $L(t)$ is a time-dependent function allow us to directly regularize $L(t)$ instead of the numerical solution $x(t)$. Here we consider regularizations of the form

$$\text{Reg}_k^p(L(\cdot)) = \int_{t_0}^{t_1} \left\| \frac{d^k L}{dt^k}(\tau) \right\|_2^p d\tau \tag{5.50}$$

with k the order derivative ($k = 0$ for no derivative) and p the type of penalization ($p = 1$ for Lasso, $p = 2$ for Ridge). The following list shows some useful penalization terms.

1. **Small angular velocities** ($k = 0$, $p = 2$). As we described in our previous result, there are many instant rotations that give rise to the same differential path. If we are interested in finding the rotation with minimum angular velocity, we can add a the penalty

$$\text{Reg}_0^2(L(\cdot)) = \int_{t_0}^{t_1} \|L(\theta)\|_2^2 d\tau. \tag{5.51}$$

   Since $\|L(t)\|_2 = \omega(t)$ coincides with the angular velocity of the infinitesimal rotation in the sphere, this penalization encourages slow rotation with large arcs (great circles) over rotations with spin axis close to the path (small circles).

2. **Smooth trajectories** ($k = 1$, $p = 2$). Smoothness in the final path can be imposed with

$$\text{Reg}_1^2(L(\cdot)) = \int_{t_0}^{t_1} \left\| \frac{dL}{dt}(\tau) \right\|_2^2 d\tau. \tag{5.52}$$

3. **Piecewise dynamics** ($k = 1$, $p = 1$). It is usally assumed that APWPs in paleomagnetism consist on a sequence of solid rotations that stay stable over certain periods of time (Gallo et al. 2022). Since constant values of $L(t)$ are associated to constant rotations with fixed rotation axis and angular velocity, we can impose sparse changes in the values of $L(t)$ by a Lasso penalty in the first derivatives of $L(t)$:

$$\text{Reg}_1^1(L(\cdot)) = \int_{t_0}^{t_1} \left\| \frac{dL}{dt}(\tau) \right\|_2 d\tau. \tag{5.53}$$

   It can be shown that this is the continuous version of the trend filtering problem (Tibshirani 2014).

Temporal uncertainties. An important component of modelling time series is how to incorporate time uncertainties and time-corrections into the model. This will assume that the observed pair (t_i, y_i) has an associated latent time τ_i and $t_i = \tau_i + \delta_i$ with δ_i independent and following some distribution including but not limited to the normal case $\delta_i \sim N(0, \sigma_i^2)$. Since time observations t_i are then subject to measurement error, we assume that directional

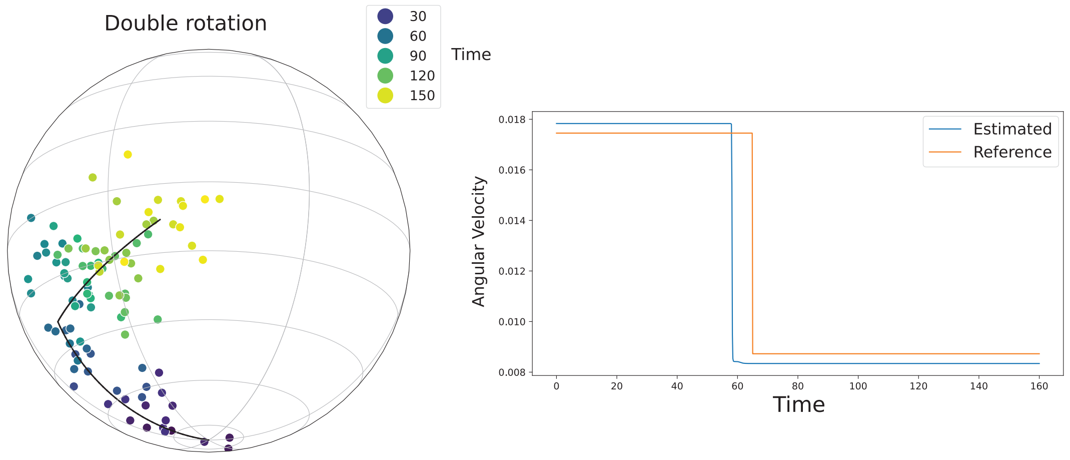


Figure 5.8: *A simple example of spherical path regression using the method introduced in this section.*

observations y_i are distributed according to $p(y|x(\tau_i), \kappa_i)$, where the latent direction $x(\tau_i)$ is evaluated at time τ_i instead of t_i. The join likelihood can then be writen as

$$p(y_i, \tau_i | x_i, t_i) = p(y_i | \tau_i, x_i)\, p(\tau_i | t_i) \tag{5.54}$$

which leads to the optimization problem

$$\min_{L(t), x_0, \tau} \sum_{i=1}^{N} -\kappa_i\, y_i^T \,\text{ODESolve}(\tau_i; x_0, L(t)) + h_i(t_i - \tau_i) \tag{5.55}$$

where $h_i : \mathbb{R} \mapsto \mathbb{R}$ is penalization term on the difference $\delta_i = t_i - \tau_i$. Different model assumptions on the temporal error will give rise to different function penalizations h_i. Two useful choices include $\delta \sim N(0, \sigma^2)$ which leads to $h(\delta) = \delta^2 / 2\sigma^2$ and $\delta \sim \text{Unif}([\delta_{\min}, \delta_{\max}])$ leading to the constrained optimization with no regularization term but restricted to $\delta \in [\delta_{\min}, \delta_{\max}]$.

The final loss function to optimize consist on

$$\min_{\theta, x_0, \tau_i} \sum_{i=1}^{N} \Big(-\kappa_i y_i^T \,\text{ODESolve}(\tau_i, x_0; L(\cdot;\theta)) + h_i(t_i - \tau_i) \Big) + \lambda\, \text{Reg}_k^p(L(\cdot)) \tag{5.56}$$

where θ are the weights and biased of the neural network parametrizing the function $L : [\text{t}_{\min}, \text{t}_{\max}] \mapsto \mathbb{R}^3$.

Figure 5.8 shows a simple example of how path regression in the sphere is carried using this method. Here we consider points distributed in the sphere generated using two different constant rotation matrices from $t_{\min} = 0$ to $t_{\max} = 160$, with the change in the rotation matrices at $t = 65$. Fisher noise with concentration parameter $\kappa = 50$ is added to the latent solution. In order to reproduce the discontinuous change in the value of $L(t)$, we added the penalization term $\text{Reg}_1^1(L(\cdot))$ to the loss function.

An implementation with examples is available via de Julia package `SphereUDE.jl` which can be directly being installed using the Julia package manager. Source code is available at `https://github.com/ODINN-SciML/SphereUDE.jl`.

Software availability. The Jupyter Notebooks and Python package created to execute the analysis in the paper is preserved at (Sapienza et al. 2023b). We also provided reproducible support by including a Binder (Jupyter et al. 2018) link to execute all the code in the cloud here `https://mybinder.org/v2/gh/PolarWandering/PaleoSampling/HEAD` and a JupyterBook (Community 2020) link here `https://polarwandering.github.io/PaleoSampling/`. We benefit from the use of PmagPy (Tauxe et al. 2016) for calculations and Dask for parallel computing (Dask Development Team 2016).

Conclusions

Pivoting between the physics and data-driven approaches, there is a unique opportunity to explore modern data science techniques to the Earth sciences. Our discussion in Chapter 1 explored some of the reason of why this is happening now and introduced some elements of physics-informed machine learning. We further emphasize the importance of designing new models that allow the assimilation based on new remote sensing observations and re-analysis products, including universal differential equations.

In Chapter 4 we introduced `ODINN.jl`, a Julia package providing high performance, source-to-source automatic differentiation and seamless integration with tools and global datasets from the Open Global Glacier Model in Python. The ice flow modelling foundations were introduced in Chapter 3. We demonstrated an application of universal differential equations as a proof of concept to learn the creep component of ice flow, i.e. a nonlinear diffusivity differential equation, of a glacier evolution model. We demonstrate this concept for 17 different glaciers around the world, for which we successfully recover a prescribed artificial law describing ice creep variability by solving approximately 500,000 ordinary differential equations in parallel.

A central element for complex data assimilation pipelines is differentiable programming. In Chapter 2, we presented a exhaustive review of the different sensitivity methods for computing gradients of loss functions that include numerical solutions of differential equations. The continuous adjoint method together with the automatic differentiation machinery allows inverse modelling inside `ODINN.jl`. Furthermore, we investigate which are the best tools in the scientific machine learning ecosystem in Julia to differentiate and optimize large nonlinear diffusivity UDEs.

We then move our discussion to paleomagnetism. In Chapter 5, we quantitatively explored, both numerically and analytically, the impact of different sampling strategies on the accuracy of estimates of paleopole position and VGP scatter. When the objective is to estimate the position of the time-averaged paleomagnetic pole, a strategy that maximizes the number of sites is always the most favorable. This research makes a step forward in introducing quantifiable metrics at the moment of estimating paleomagnetic pole position and dispersion in paleomagnetic studies. We also emphasize that beyond these general rules of thumb, we herein provided tools enabling quantitative sampling recommendations to be generated from user-provided expectations. To conclude, we further shown how universal differential equations can be used for estimation of paleopole trajectories on the sphere.

Bibliography

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng (2016). “TensorFlow: A System for Large-Scale Machine Learning”. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI’16. Savannah, GA, USA: USENIX Association, pp. 265–283.

Abdalati, W., H. J. Zwally, R. Bindschadler, B. Csatho, S. L. Farrell, H. A. Fricker, D. Harding, R. Kwok, M. Lefsky, T. Markus, et al. (2010). “The ICESat-2 laser altimetry mission”. In: *Proceedings of the IEEE* 98.5, pp. 735–751.

Abernathey, R. P., T. Augspurger, A. Banihirwe, C. C. Blackmon-Luca, T. J. Crone, C. L. Gentemann, J. J. Hamman, N. Henderson, C. Lepore, T. A. McCaie, N. H. Robinson, and R. P. Signell (2021). “Cloud-Native Repositories for Big Scientific Data”. In: *Computing in Science Engineering* 23.2, pp. 26–35. DOI: 10.1109/MCSE.2021.3059437.

Alexe, M. and A. Sandu (2007). “DENSERKS: Fortran sensitivity solvers using continuous, explicit Runge-Kutta schemes”. In.

— (2009). “Forward and adjoint sensitivity analysis with continuous explicit Runge–Kutta schemes”. In: *Applied Mathematics and Computation* 208.2, pp. 328–346. DOI: 10.1016/j.amc.2008.11.035.

Allaire, G., C. Dapogny, and P. Frey (2014). “Shape optimization with a level set based mesh evolution method”. In: *Computer Methods in Applied Mechanics and Engineering* 282, pp. 22–53.

Anilkumar, R., R. Bharti, D. Chutia, and S. P. Aggarwal (2022). “Modelling the Point Mass Balance for the Glaciers of Central European Alps using Machine Learning Techniques”. In: *EGUsphere*, pp. 1–27.

Arakawa, A. and V. R. Lamb (1977). “Computational design of the basic dynamical processes of the UCLA general circulation model”. In: *General circulation models of the atmosphere* 17.Supplement C, pp. 173–265.

Arendt, A. A., J. Hamman, M. Rocklin, A. Tan, D. R. Fatland, J. Joughin, E. D. Gutmann, L. Setiawan, and S. T. Henderson (2018). “Pangeo: Community tools for analysis of Earth Science Data in the Cloud”. In: *AGU Fall Meeting Abstracts*. Vol. 2018, IN54A–05.

Arthern, R. J. and G. H. Gudmundsson (2010). “Initialization of ice-sheet forecasts viewed as an inverse Robin problem”. en. In: *Journal of Glaciology* 56.197, pp. 527–533. DOI: 10.3189/002214310792447699.

Ascher, U. M. (2008). *Numerical methods for evolutionary differential equations*. SIAM.

Ascher, U. M. and C. Greif (2011). *A First Course in Numerical Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics. DOI: 10.1137/9780898719987.

Azari, A. R., E. Abrahams, F. Sapienza, D. L. Mitchell, J. Biersteker, S. Xu, C. Bowers, F. Pérez, G. A. DiBraccio, Y. Dong, and S. Curry (2023). “Magnetic Field Draping in Induced Magnetospheres: Evidence From the MAVEN Mission to Mars”. In: *Journal of Geophysical Research: Space Physics* 128.11, e2023JA031546. DOI: https://doi.org/10.1029/2023JA031546.

Azari, A. R., J. W. Lockhart, M. W. Liemohn, and X. Jia (2020). “Incorporating Physical Knowledge Into Machine Learning for Planetary Space Physics”. In: *Frontiers in Astronomy and Space Sciences* 7, p. 36. DOI: 10.3389/fspas.2020.00036.

Azari, A., E. Abrahams, F. Sapienza, J. Halekas, J. Biersteker, D. Mitchell, F. Pérez, M. Marquette, M. Rutala, C. Bowers, et al. (2024). “A Virtual Solar Wind Monitor for Mars with Uncertainty Quantification using Gaussian Processes”. In: *arXiv preprint arXiv:2402.01932*.

Barton, R. R. (1992). “Computing Forward Difference Derivatives In Engineering Optimization”. In: *Engineering Optimization* 20.3, pp. 205–224. DOI: 10.1080/03052159208941281.

Bauer, F. L. (1974). “Computational Graphs and Rounding Error”. In: *SIAM Journal on Numerical Analysis* 11.1, pp. 87–96. DOI: 10.1137/0711010.

Baumhoer, C. A., A. J. Dietz, C. Kneisel, and C. Kuenzer (Oct. 2019). “Automated Extraction of Antarctic Glacier and Ice Shelf Fronts from Sentinel-1 Imagery Using Deep Learning”. en. In: *Remote Sensing* 11.21, p. 2529. DOI: 10.3390/rs11212529.

Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (Jan. 2017). “Automatic Differentiation in Machine Learning: A Survey”. In: *J. Mach. Learn. Res.* 18.1, pp. 5595–5637.

Behn, M. D., D. L. Goldsby, and G. Hirth (2021). “The role of grain size evolution in the rheology of ice: implications for reconciling laboratory creep data and the Glen flow law”. In: *The Cryosphere* 15.9, pp. 4589–4605. DOI: 10.5194/tc-15-4589-2021.

Bennett, C. H. (1973). “Logical Reversibility of Computation”. In: *IBM Journal of Research and Development* 17.6, pp. 525–532. DOI: 10.1147/rd.176.0525.

Berlinghieri, R., B. L. Trippe, D. R. Burt, R. Giordano, K. Srinivasan, T. Özgökmen, J. Xia, and T. Broderick (2023). “Gaussian processes at the Helm(holtz): A more fluid model for ocean currents”. In: *arXiv*. DOI: 10.48550/arxiv.2302.10364.

Besse, J. and V. Courtillot (2002). “Apparent and true polar wander and the geometry of the geomagnetic field over the last 200 Myr”. In: *Journal of Geophysical Research: Solid Earth* 107.B11, EPM–6.

Betancourt, M. (2017). “A Conceptual Introduction to Hamiltonian Monte Carlo”. In: *arXiv*. DOI: 10.48550/arxiv.1701.02434.

Bettencourt, J., M. J. Johnson, and D. Duvenaud (2019). "Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX". In: *Program Transformations for ML Workshop at NeurIPS 2019*.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1, pp. 65–98. DOI: 10.1137/141000671.

Bezanson, J., S. Karpinski, V. B. Shah, and A. Edelman (2012). "Julia: A Fast Dynamic Language for Technical Computing". In: *arXiv*. DOI: 10.48550/arxiv.1209.5145.

Blondel, M. and V. Roulet (2024). "The Elements of Differentiable Programming". In: *arXiv*.

Bolibar, J., F. Sapienza, F. Maussion, R. Lguensat, B. Wouters, and F. Pérez (2023a). "Universal differential equations for glacier ice flow modelling". In: *Geoscientific Model Development* 16.22, pp. 6671–6687. DOI: 10.5194/gmd-16-6671-2023.

Bolibar, J., A. Rabatel, I. Gouttevin, and C. Galiez (Sept. 2020a). "A deep learning reconstruction of mass balance series for all glaciers in the French Alps: 1967–2015". en. In: *Earth System Science Data* 12.3, pp. 1973–1983. DOI: 10.5194/essd-12-1973-2020.

Bolibar, J., A. Rabatel, I. Gouttevin, C. Galiez, T. Condom, and E. Sauquet (Feb. 2020b). "Deep learning applied to glacier evolution modelling". en. In: *The Cryosphere* 14.2, pp. 565–584. DOI: 10.5194/tc-14-565-2020.

Bolibar, J., A. Rabatel, I. Gouttevin, H. Zekollari, and C. Galiez (Dec. 2022). "Nonlinear sensitivity of glacier mass balance to future climate change unveiled by deep learning". en. In: *Nature Communications* 13.1, p. 409. DOI: 10.1038/s41467-022-28033-0.

Bolibar, J. and F. Sapienza (June 2023b). *ODINN-SciML/ODINN.jl: v0.2.0*. Version v0.2.0. DOI: 10.5281/zenodo.8033313.

Bradbury, J., R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne (2020). "JAX: composable transformations of Python+ NumPy programs, 2018". In: *URL http://github. com/google/jax*, p. 18.

Bradley, A. M. (2013). *PDE-constrained optimization and the adjoint method*. Tech. rep. Technical Report. Stanford University. https://cs. stanford. edu/˜ ambrad ...

Breiman, L. (2001). "Statistical modeling: The two cultures (with comments and a rejoinder by the author)". In: *Statistical science* 16.3, pp. 199–231.

Brenner, H. (2005). "Navier–Stokes revisited". In: *Physica A: Statistical Mechanics and its Applications* 349.1–2, pp. 60–132. DOI: 10.1016/j.physa.2004.10.034.

Brinkerhoff, D., A. Aschwanden, and M. Fahnestock (June 2020). "Constraining subglacial processes from surface velocity observations using surrogate-based Bayesian inference". In: *arXiv:2006.12422 [physics]*. arXiv: 2006.12422.

Brinkerhoff, D. J., C. R. Meyer, E. Bueler, M. Truffer, and T. C. Bartholomaus (2016). "Inversion of a glacier hydrology model". In: *Annals of Glaciology* 57.72, pp. 84–95. DOI: 10.1017/aog.2016.3.

Brooks Jr, F. P. (1995). *The mythical man-month (anniversary ed.)*

Brunton, S. L., J. L. Proctor, and J. N. Kutz (Apr. 2016). "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". en. In: *Proceedings* *of the National Academy of Sciences* 113.15, pp. 3932–3937. DOI: 10.1073/pnas.1517384113.

Bryson, A., Y.-C. Ho, and G. Siouris (July 1979). “Applied Optimal Control: Optimization, Estimation, and Control”. In: *Systems, Man and Cybernetics, IEEE Transactions on* 9, pp. 366–367. DOI: 10.1109/TSMC.1979.4310229.

Bueler, E. (2014). “Numerical Modelling of Ice Sheets, Streams, and Shelves”. In: *Karthaus*.

Bueler, E. and J. Brown (2009). “Shallow shelf approximation as a “sliding law” in a thermomechanically coupled ice sheet model”. In: *Journal of Geophysical Research: Earth Surface (2003–2012)* 114.F3. DOI: 10.1029/2008jf001179.

Bui-Thanh, T., C. Burstedde, O. Ghattas, J. Martin, G. Stadler, and L. C. Wilcox (2012). “Extreme-scale UQ for Bayesian inverse problems governed by PDEs”. In: *IEEE Computer Society Press*, p. 3.

Buss, S. R. and J. P. Fillmore (2001). “Spherical averages and applications to spherical splines and interpolation”. In: *ACM Transactions on Graphics (TOG)* 20.2, pp. 95–126. DOI: 10.1145/502122.502124.

Butcher, J. C. (2001). “Numerical Analysis: Historical Developments in the 20th Century”. In: pp. 449–477. DOI: 10.1016/b978-0-444-50617-7.50018-5.

Butcher, J. and G. Wanner (1996). “Runge-Kutta methods: some historical notes”. In: *Applied Numerical Mathematics* 22.1–3, pp. 113–151. DOI: 10.1016/s0168-9274(96)00048-7.

Butler, R. F. (1992). *Paleomagnetism: magnetic domains to geologic terranes*. Vol. 319. Blackwell Scientific Publications Boston.

Cao, Y., S. Li, and L. Petzold (2002). “Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software”. In: *Journal of Computational and Applied Mathematics* 149.1, pp. 171–191. DOI: 10.1016/s0377-0427(02)00528-9.

Cerisola, F., F. Sapienza, and A. J. Roncaglia (2022). “Heat engines with single-shot deterministic work extraction”. In: *Physical Review E* 106.3, p. 034135.

Chanussot, L., A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, A. Palizhati, A. Sriram, B. Wood, J. Yoon, D. Parikh, C. L. Zitnick, and Z. Ulissi (2021). “Open Catalyst 2020 (OC20) Dataset and Community Challenges”. In: *ACS Catalysis*. DOI: 10.1021/acscatal.0c04525.

Chazal, F., L. Ferraris, P. Groisman, M. Jonckheere, F. Pascal, and F. Sapienza (2023). “Choosing the parameter of the Fermat distance: navigating geometry and noise”. In: *arXiv preprint arXiv:2311.18663*.

Chen, R., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud (2018). “Neural ordinary differential equations”. In: *Advances in neural information processing systems* 31.

Chen, Y., B. Hosseini, H. Owhadi, and A. M. Stuart (2021). “Solving and Learning Nonlinear PDEs with Gaussian Processes”. In: *arXiv*. DOI: 10.48550/arxiv.2103.12959.

Chen, Y., Y. Luo, Q. Liu, H. Xu, and D. Zhang (2022). “Symbolic genetic algorithm for discovering open-form partial differential equations (SGA-PDE)”. In: *Physical Review Research* 4.2, p. 023174. DOI: 10.1103/physrevresearch.4.023174.

Chiles, J.-P. and P. Delfiner (2012). *Geostatistics: modeling spatial uncertainty*. Vol. 713. John Wiley & Sons.

Cleary, E., A. Garbuno-Inigo, S. Lan, T. Schneider, and A. M. Stuart (2021). "Calibrate, emulate, sample". In: *Journal of Computational Physics* 424, p. 109716. DOI: 10.1016/j.jcp.2020.109716.

Clifford (1871). "Preliminary sketch of biquaternions". In: *Proceedings of the London Mathematical Society* 1.1, pp. 381–395.

Community, E. B. (Feb. 2020). *Jupyter Book*. Version v0.10. DOI: 10.5281/zenodo.4539666.

Consortium, G. (2019). *Glacier Thickness Database 3.0.1*.

Consortium, R. G. I. (2017). *Randolph Glacier Inventory 6.0*. English. type: dataset. DOI: 10.7265/N5-RGI-60.

Constable, C. and R. Parker (1988). "Statistics of the geomagnetic secular variation for the past 5 my". In: *Journal of Geophysical Research: Solid Earth* 93.B10, pp. 11569–11581.

Coveney, P. V., E. R. Dougherty, and R. R. Highfield (2016). "Big data need big theory too". In: *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences* 374.2080, pp. 20160153–11. DOI: 10.1098/rsta.2016.0153.

Cox, A. (1970). "Latitude dependence of the angular dispersion of the geomagnetic field". In: *Geophysical Journal International* 20.3, pp. 253–269.

Cranmer, K., J. Brehmer, and G. Louppe (2020). "The frontier of simulation-based inference." In: *Proceedings of the National Academy of Sciences of the United States of America* 117.48, pp. 30055–30062. DOI: 10.1073/pnas.1912789117.

Creer, K., E. Irving, and S. Runcorn (1954). "The direction of the geomagnetic field in remote epochs in Great Britain". In: *Journal of geomagnetism and geoelectricity* 6.4, pp. 163–168.

Creswell, R., K. M. Shepherd, B. Lambert, G. R. Mirams, C. L. Lei, S. Tavener, M. Robinson, and D. J. Gavaghan (2023). "Understanding the impact of numerical solvers on inference for differential equation models". In: *arXiv preprint arXiv:2307.00749*.

Cromwell, G., C. Johnson, L. Tauxe, C. Constable, and N. Jarboe (2018). "PSV10: A global data set for 0–10 Ma time-averaged field and paleosecular variation studies". In: *Geochemistry, Geophysics, Geosystems* 19.5, pp. 1533–1558.

Cuffey, K. and W. S. B. Paterson (2010). *The physics of glaciers*. 4th ed. OCLC: ocn488732494. Burlington, MA: Butterworth-Heinemann/Elsevier.

Cuoco, E., J. Powell, M. Cavaglià, K. Ackley, M. Bejger, C. Chatterjee, M. Coughlin, S. Coughlin, P. Easter, R. Essick, H. Gabbard, T. Gebhard, S. Ghosh, L. Haegel, A. Iess, D. Keitel, Z. Marka, S. Marka, F. Morawski, T. Nguyen, R. Ormiston, M. Puerrer, M. Razzano, K. Staats, G. Vajente, and D. Williams (2020). "Enhancing Gravitational-Wave Science with Machine Learning". In: *arXiv*. DOI: 10.1088/2632-2153/abb93a.

Dahlquist, G. (1985). "33 years of numerical instability, Part I". In: *BIT Numerical Mathematics* 25.1, pp. 188–204. DOI: 10.1007/bf01934997.

Dandekar, R., K. Chung, V. Dixit, M. Tarek, A. Garcia-Valadez, K. V. Vemula, and C. Rackauckas (2020). "Bayesian Neural Ordinary Differential Equations". In: *arXiv*.

Dask Development Team (2016). *Dask: Library for dynamic task scheduling*.

Deenen, M. H., C. G. Langereis, D. J. van Hinsbergen, and A. J. Biggin (2011). “Geomagnetic secular variation and the statistics of palaeomagnetic directions”. In: *Geophysical Journal International* 186.2, pp. 509–520.

Doubrovine, P. V., T. Veikkolainen, L. J. Pesonen, E. Piispa, S. Ots, A. V. Smirnov, E. V. Kulakov, and A. J. Biggin (2019). “Latitude dependence of geomagnetic paleosecular variation and its relation to the frequency of magnetic reversals: observations from the Cretaceous and Jurassic”. In: *Geochemistry, Geophysics, Geosystems* 20.3, pp. 1240–1279.

Dürrbaum, A., W. Klier, and H. Hahn (2002). “Comparison of Automatic and Symbolic Differentiation in Mathematical Modeling and Computer Simulation of Rigid-Body Systems”. In: *Multibody System Dynamics* 7.4, pp. 331–355. DOI: 10.1023/a:1015523018029.

Eberhard, P. and C. Bischof (1996). “Automatic differentiation of numerical integration algorithms”. In: *Mathematics of Computation* 68.226, pp. 717–731. DOI: 10.1090/s0025-5718-99-01027-3.

Ebert-Uphoff, I. and Y. Deng (2014). “Causal Discovery from Spatio-Temporal Data with Applications to Climate Science”. In: *2014 13th International Conference on Machine Learning and Applications*, pp. 606–613. DOI: 10.1109/icmla.2014.96.

Elliott, C. (2018). “The simple essence of automatic differentiation”. In: *Proceedings of the ACM on Programming Languages* 2.ICFP, p. 70. DOI: 10.1145/3236765.

Elliott, J. and J. Peraire (1996). “Aerodynamic design using unstructured meshes”. In: *Fluid Dynamics Conference*. DOI: 10.2514/6.1996-1941.

Elvira, V. D., S. Gottlieb, O. Gutsche, B. Nachman, S. Bailey, W. Bhimji, P. Boyle, G. Cerati, M. C. Kind, K. Cranmer, et al. (2022). “The Future of High Energy Physics Software and Computing”. In: *arXiv preprint arXiv:2210.05822*.

Farinotti, D., M. Huss, J. J. Fürst, J. Landmann, H. Machguth, F. Maussion, and A. Pandit (Mar. 2019). “A consensus estimate for the ice thickness distribution of all glaciers on Earth”. en. In: *Nature Geoscience* 12.3, pp. 168–173. DOI: 10.1038/s41561-019-0300-3.

Farr, T. G., P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, et al. (2007). “The shuttle radar topography mission”. In: *Reviews of geophysics* 45.2.

Farrell, P. E., D. A. Ham, S. W. Funke, and M. E. Rognes (2013). “Automated Derivation of the Adjoint of High-Level Transient Finite Element Programs”. In: *SIAM Journal on Scientific Computing* 35.4, pp. C369–C393. DOI: 10.1137/120873558.

Fike, J. A. (2013). “Multi-objective optimization using hyper-dual numbers”. PhD thesis.

Fisher, R. A. (1953). “Dispersion on a sphere”. In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 217, pp. 295–305.

Fletcher, P. T. (2013). “Geodesic Regression and the Theory of Least Squares on Riemannian Manifolds”. In: *International Journal of Computer Vision* 105.2, pp. 171–185. DOI: 10.1007/s11263-012-0591-y.

Fornberg, B. (1988). “Generation of Finite Difference Formulas on Arbitrarily Spaced Grids”. In: *Mathematics of Computation* 51.184, pp. 699–706.

Fowler, A. (2010). “Weertman, Lliboutry and the development of sliding theory”. In: *Journal of Glaciology* 56.200, pp. 965–972. DOI: 10.3189/002214311796406112.

Fowler, A. and F. Ng (2020). *Glaciers and Ice Sheets in the climate system: The Karthaus summer school lecture notes*. Springer Nature.

Gallo, L. C., M. Domeier, F. Sapienza, N. L. Swanson-Hysell, B. Vaes, Y. Zhang, M. Arnould, A. Eyster, D. Gürer, Á. Király, B. Robert, T. Rolf, G. Shephard, and A. van der Boon (2023). “Embracing Uncertainty to Resolve Polar Wander: A Case Study of Cenozoic North America”. In: *Geophysical Research Letters* 50.11. DOI: 10.1029/2023gl103436.

Gallo, L. C., F. Sapienza, and M. Domeier (2022). “An optimization method for paleomagnetic Euler pole analysis”. In: *Computers & Geosciences* 166, p. 105150.

Gallo, L. C. et al. (2024). “On the feasibility of paleomagnetic Euler pole analysis”. In: *preparation*.

Gebremedhin, A. H., F. Manne, and A. Pothen (2005). “What color is your Jacobian? Graph coloring for computing derivatives”. In: *SIAM review* 47.4, pp. 629–705.

Gelbrecht, M., A. White, S. Bathiany, and N. Boers (2023). “Differentiable programming for Earth system modeling”. In: *Geoscientific Model Development* 16.11, pp. 3123–3135. DOI: 10.5194/gmd-16-3123-2023.

Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin (2013). *Bayesian data analysis*. CRC press.

Gentemann, C. L., C. Holdgraf, R. Abernathey, D. Crichton, J. Colliander, E. J. Kearns, Y. Panda, and R. P. Signell (2021). “Science Storms the Cloud”. In: *AGU Advances* 2.2. DOI: 10.1029/2020av000354.

Gerritsen, D., B. Vaes, and D. J. van Hinsbergen (2022). “Influence of data filters on the position and precision of paleomagnetic poles: what is the optimal sampling strategy?” In: *Geochemistry, Geophysics, Geosystems* 23.4, e2021GC010269.

Ghattas, O. and K. Willcox (2021). “Learning physics-based models from data: perspectives from inverse problems and model reduction”. In: *Acta Numerica* 30, pp. 445–554. DOI: 10.1017/s0962492921000064.

Giering, R. and T. Kaminski (1998). “Recipes for adjoint code construction”. In: *ACM Transactions on Mathematical Software (TOMS)* 24.4, pp. 437–474. DOI: 10.1145/293686.293695.

Giles, M. B. and N. A. Pierce (2000a). “An introduction to the adjoint approach to design”. In: *Flow, Turbulence and Combustion* 65.3, pp. 393–415.

Giles, M. B. and N. A. Pierce (2000b). “An Introduction to the Adjoint Approach to Design”. In: *Flow, Turbulence and Combustion* 65.3–4, pp. 393–415. DOI: 10.1023/a:1011430410075.

Givoli, D. (2021). “A tutorial on the adjoint method for inverse problems”. In: 380, p. 113810. DOI: 10.1016/j.cma.2021.113810.

Glen, J. W. (1955). “The creep of polycrystalline ice”. In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 228.1175, pp. 519–538.

Goldberg, D. N. and P. Heimbach (2013). “Parameter and state estimation with a time-dependent adjoint marine ice sheet model”. In: *The Cryosphere* 7.6, pp. 1659–1678. DOI: 10.5194/tc-7-1659-2013.

Goldberg, D. (1991). “What every computer scientist should know about floating-point arithmetic”. In: *ACM Computing Surveys (CSUR)* 23.1, pp. 5–48. DOI: 10.1145/103162.103163.

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

Gowda, S., Y. Ma, A. Cheli, M. Gwóźzdź, V. B. Shah, A. Edelman, and C. Rackauckas (2022). “High-performance symbolic-numerics via multiple dispatch”. In: *ACM Communications in Computer Algebra* 55.3, pp. 92–96. DOI: 10.1145/3511528.3511535.

Granger, B. E. and F. Pérez (2021). “Jupyter: Thinking and Storytelling With Code and Data”. In: *Computing in Science Engineering* 23.2, pp. 7–14. DOI: 10.1109/mcse.2021.3059263.

Green, P. J. and B. W. Silverman (1993). *Nonparametric regression and generalized linear models: a roughness penalty approach*. Crc Press.

Griewank, A. (1989). “On Automatic Differentiation”. In: *Mathematical Programming: Recent Developments and Applications*.

— (2012). “Who invented the reverse mode of differentiation”. In: *Documenta Mathematica, Extra Volume ISMP* 389400.

Griewank, A. and A. Walther (2008). *Evaluating Derivatives*. DOI: 10.1137/1.9780898717761.

Gronwall, T. H. (1919). “Note on the derivatives with respect to a parameter of the solutions of a system of differential equations”. In: *Annals of Mathematics*, pp. 292–296.

Guidicelli, M., M. Huss, M. Gabella, and N. Salzmann (2023). “Spatio-temporal reconstruction of winter glacier mass balance in the Alps, Scandinavia, Central Asia and western Canada (1981–2019) using climate reanalyses and machine learning”. In: *The Cryosphere* 17.2, pp. 977–1002.

Hager, W. W. (2000). “Runge-Kutta methods in optimal control and the transformed adjoint system”. In: *Numerische Mathematik* 87.2, pp. 247–282. DOI: 10.1007/s002110000178.

Hairer, E., G. Wanner, and S. Nørsett (2008). *Solving Ordinary Differential Equations I: Nonstiff Problems (Second Revised Edition)*. Springer Berlin Heidelberg New York.

Halevy, A., P. Norvig, and F. Pereira (2009). “The Unreasonable Effectiveness of Data”. In: *IEEE Intelligent Systems* 24.2, pp. 8–12. DOI: 10.1109/mis.2009.36.

Halfar, P. (1981). “On the dynamics of the ice sheets”. In: *Journal of Geophysical Research: Oceans* 86.C11, pp. 11065–11072. DOI: 10.1029/jc086ic11p11065.

Han, J., A. Jentzen, and W. E (2018). “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences* 115.34, p. 201718942. DOI: 10.1073/pnas.1718942115.

Harris, C. R., K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. (2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362.

Hasani, E. and R. A. Ward (2024). "Generating synthetic data for neural operators". In: *arXiv*.

Hascoët, L. and M. Morlighem (2018). "Source-to-source adjoint Algorithmic Differentiation of an ice sheet model written in C". In: *Optimization Methods and Software* 33.4-6, pp. 829–843.

Hastie, T., R. Tibshirani, J. H. Friedman, and J. H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

He, Q., M. Perego, A. A. Howard, G. E. Karniadakis, and P. Stinis (2023). "A hybrid deep neural operator/finite element method for ice-sheet modeling". In: *Journal of Computational Physics* 492, p. 112428. DOI: 10.1016/j.jcp.2023.112428.

Heimbach, P. and V. Bugnion (2009). "Greenland ice-sheet volume sensitivity to basal, surface and initial conditions derived from an adjoint model". In: *Annals of Glaciology* 50.52, pp. 67–80.

Heinonen, M., C. Yildiz, H. Mannerström, J. Intosalmi, and H. Lähdesmäki (2018). "Learning unknown ODE models with Gaussian processes". In: *arXiv*.

Hermans, J., V. Begy, and G. Louppe (2020). "Likelihood-free mcmc with amortized approximate ratio estimators". In: *International conference on machine learning*. PMLR, pp. 4239–4248.

Heslop, D. and A. Roberts (2020). "Uncertainty propagation in hierarchical paleomagnetic reconstructions". In: *Journal of Geophysical Research: Solid Earth* 125.6, e2020JB019488.

Hey, T., S. Tansley, K. Tolle, and J. Gray (Oct. 2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396.

Hock, R. (Nov. 2003). "Temperature index melt modelling in mountain areas". en. In: *Journal of Hydrology* 282.1-4, pp. 104–115. DOI: 10.1016/S0022-1694(03)00257-9.

Hock, R., F. Maussion, B. Marzeion, and S. Nowicki (2023). "What is the global glacier ice volume outside the ice sheets?" In: *Journal of Glaciology* 69.273, pp. 204–210. DOI: 10.1017/jog.2023.1.

Hoffimann, J., M. Zortea, B. d. Carvalho, and B. Zadrozny (2021). "Geostatistical Learning: Challenges and Opportunities". In: *arXiv*.

Hoyer, S. and J. Hamman (2017a). "xarray: N-D labeled arrays and datasets in Python". In: *Journal of Open Research Software* 5.1. DOI: 10.5334/jors.148.

Hoyer, S. and J. J. Hamman (Apr. 2017b). "xarray: N-D labeled Arrays and Datasets in Python". en. In: *Journal of Open Research Software* 5, p. 10. DOI: 10.5334/jors.148.

Huang, J., T. M. Smith, G. M. Henry, and R. A. V. D. Geijn (2016). "Strassen's Algorithm Reloaded". In: *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 690–701. DOI: 10.1109/sc.2016.58.

Hugonnet, R., R. McNabb, E. Berthier, B. Menounos, C. Nuth, L. Girod, D. Farinotti, M. Huss, I. Dussaillant, F. Brun, and A. Kääb (Mar. 2020). *A globally complete, spatially and temporally resolved estimate of glacier mass change: 2000 to 2019*. other. display. DOI: 10.5194/egusphere-egu2020-20908.

Hunter, J. D. (2007). "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Hüper, K. and F. S. Leite (2007). "On the Geometry of Rolling and Interpolation Curves on Sn, SOn, and Grassmann Manifolds". In: *Journal of Dynamical and Control Systems* 13.4, pp. 467–502. DOI: 10.1007/s10883-007-9027-3.

Huss, M. and R. Hock (Sept. 2015). "A new model for global glacier change and sea-level rise". In: *Frontiers in Earth Science* 3. DOI: 10.3389/feart.2015.00054.

Hutter, K. (1983). *Theoretical Glaciology*. en. Dordrecht: Springer Netherlands. DOI: 10.1007/978-94-015-1167-4.

Imhof, M. A. (2021). "Combined climate-ice flow modelling of the Alpine ice field during the Last Glacial Maximum". In: *VAW-Mitteilungen* 260.

Innes, M. (2018). "Don't Unroll Adjoint: Differentiating SSA-Form Programs". In: *arXiv*.

Innes, M., E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah (2018). "Fashionable Modelling with Flux". In: *CoRR* abs/1811.01457.

Innes, M., A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt (2019). "A Differentiable Programming System to Bridge Machine Learning and Scientific Computing". In: *arXiv*. DOI: 10.48550/arxiv.1907.07587.

Ipsen, I. C. F. and C. D. Meyer (1998). "The Idea Behind Krylov Methods". In: *The American Mathematical Monthly* 105.10, pp. 889–899. DOI: 10.1080/00029890.1998.12004985.

Irving, E. (1977). "Drift of the major continental blocks since the Devonian". In: *Nature* 270.5635, pp. 304–309.

Iten, R., T. Metger, H. Wilming, L. d. Rio, and R. Renner (2020). "Discovering Physical Concepts with Neural Networks". In: *Physical Review Letters* 124.1, p. 010508. DOI: 10.1103/physrevlett.124.010508.

Jain, A., S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, et al. (2013). "Commentary: The Materials Project: A materials genome approach to accelerating materials innovation". In: *APL materials* 1.1.

Jameson, A. (1988). "Aerodynamic design via control theory". In: *Journal of Scientific Computing* 3.3, pp. 233–260. DOI: 10.1007/bf01061285.

Jensen, J. S., P. B. Nakshatrala, and D. A. Tortorelli (2014). "On the consistency of adjoint sensitivity analysis for structural optimization of linear dynamic problems". In: *Structural and Multidisciplinary Optimization* 49.5, pp. 831–837. DOI: 10.1007/s00158-013-1024-4.

Johnson, S. G. (2012). "Notes on Adjoint Methods for 18.335". In.

Jouvet, G. (2023). “Inversion of a Stokes glacier flow model emulated by deep learning”. In: *Journal of Glaciology* 69.273, pp. 13–26. DOI: 10.1017/jog.2022.41.

Jouvet, G., G. Cordonnier, B. Kim, M. Lüthi, A. Vieli, and A. Aschwanden (Dec. 2021). “Deep learning speeds up ice flow modelling by several orders of magnitude”. en. In: *Journal of Glaciology*, pp. 1–14. DOI: 10.1017/jog.2021.120.

Juedes, D. W. (1991). *A taxonomy of automatic differentiation tools*. Tech. rep. Argonne National Lab., IL (United States).

Jupp, P. E. and J. T. Kent (1987). “Fitting Smooth Paths to Speherical Data”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 36.1, pp. 34–46.

Jupyter, P., M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing (2018). “Binder 2.0 - Reproducible, interactive, sharable environments for science at scale”. In: *Proceedings of the 17th Python in Science Conference*. Ed. by F. Akici, D. Lippa, D. Niederhut, and M. Pacer, pp. 113–120. DOI: 10.25080/Majora-4af1f417-011.

Kaltenborn, J., C. E. E. Lange, V. Ramesh, P. Brouillard, Y. Gurwicz, C. Nagda, J. Runge, P. Nowack, and D. Rolnick (2023). “ClimateSet: A Large-Scale Climate Model Dataset for Machine Learning”. In: *arXiv*. DOI: 10.48550/arxiv.2311.03721.

Kantorovich, L. V. (1957). “On a mathematical symbolism convenient for performing machine calculations”. In: *Dokl. Akad. Nauk SSSR*. Vol. 113. 4, pp. 738–741.

Karczmarczuk, J. (1998). “Functional Differentiation of Computer Programs”. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*. ICFP ’98. Baltimore, Maryland, USA: Association for Computing Machinery, pp. 195–203. DOI: 10.1145/289423.289442.

Karniadakis, G. E., I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang (2021). “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6, pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

Karpatne, A., G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar (2017a). “Theory-Guided Data Science: A New Paradigm for Scientific Discovery from Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.10, pp. 2318–2331. DOI: 10.1109/tkde.2017.2720168.

Karpatne, A., I. Ebert-Uphoff, S. Ravela, H. A. Babaie, and V. Kumar (2017b). “Machine Learning for the Geosciences: Challenges and Opportunities”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.8, pp. 1544–1554. DOI: 10.1109/tkde.2018.2861006.

Kent, D. V. and E. Irving (2010). “Influence of inclination error in sedimentary rocks on the Triassic and Jurassic apparent pole wander path for North America and implications for Cordilleran tectonics”. In: *Journal of Geophysical Research: Solid Earth* 115.B10.

Khokhlov, A. and G. Hulot (Jan. 2016). “Principal component analysis of palaeomagnetic directions: converting a Maximum Angular Deviation (MAD) into an 95 angle”. In: *Geophysical Journal International* 204.1, pp. 274–291. DOI: 10.1093/gji/ggv451.

Kidger, P. (2021). “On Neural Differential Equations”. PhD thesis. University of Oxford.

Kidger, P., R. T. Q. Chen, and T. J. Lyons (18–24 Jul 2021). “"Hey, that’s not an ODE": Faster ODE Adjoints via Seminorms”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 5443–5452.

Kim, S., W. Ji, S. Deng, Y. Ma, and C. Rackauckas (2021). “Stiff neural ordinary differential equations”. en. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.9, p. 093122. DOI: 10.1063/5.0060697.

Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. development team (2016). “Jupyter Notebooks - a publishing format for reproducible computational workflows”. In: ed. by F. Loizides and B. Scmidt, pp. 87–90.

Kochkov, D., J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer (2021). “Machine learning accelerated computational fluid dynamics”. In: *arXiv*.

Kovachki, N., Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar (2021). “Neural Operator: Learning Maps Between Function Spaces”. In: *arXiv*. DOI: 10.48550/arxiv.2108.08481.

Kuhn, T. S. (1962). *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press.

Kurz, G., F. Pfaff, and U. D. Hanebeck (2016). “Kullback-Leibler divergence and moment matching for hyperspherical probability distributions”. In: *2016 19th International Conference on Information Fusion (FUSION)*. IEEE, pp. 2087–2094.

Lagergren, J. H., J. T. Nardini, R. E. Baker, M. J. Simpson, and K. B. Flores (2020). “Biologically-informed neural networks guide mechanistic modeling from sparse experimental data”. In: *arXiv*. DOI: 10.1371/journal.pcbi.1008462.

Lam, R., A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia (2022). “GraphCast: Learning skillful medium-range global weather forecasting”. In: *arXiv*. DOI: 10.48550/arxiv.2212.12794.

Landau, L. D. and E. M. Lifshitz (2013). *Fluid Mechanics: Landau and Lifshitz: Course of Theoretical Physics, Volume 6*. Vol. 6. Elsevier.

Lange, S. (2019). *WFDE5 over land merged with ERA5 over the ocean (W5E5)*. en. Version Number: 1.0 Type: dataset. DOI: 10.5880/PIK.2019.023.

Lantoine, G., R. P. Russell, and T. Dargent (2012). “Using Multicomplex Variables for Automatic Computation of High-Order Derivatives”. In: *ACM Transactions on Mathematical Software (TOMS)* 38.3, p. 16. DOI: 10.1145/2168773.2168774.

Laue, S. (2019). *On the Equivalence of Forward Mode Automatic Differentiation and Symbolic Differentiation*. DOI: 10.48550/ARXIV.1904.02990.

LeCun, Y., Y. Bengio, and G. Hinton (May 27, 2015). “Deep Learning”. In: *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539.

Leis, J. R. and M. A. Kramer (Mar. 1988). “Algorithm 658: ODESSA–an Ordinary Differential Equation Solver with Explicit Simultaneous Sensitivity Analysis”. In: *ACM Trans. Math. Softw.* 14.1, pp. 61–67. DOI: 10.1145/42288.214371.

Lemarié-Rieusset, P. G. (2018). *The Navier-Stokes problem in the 21st century*. CRC press.

Leong, W. J. and H. J. Horgan (Apr. 2020). *DeepBedMap: Using a deep neural network to better resolve the bed topography of Antarctica*. preprint. Ice sheets/Data Assimilation. DOI: 10.5194/tc-2020-74.

Lguensat, R., J. L. Sommer, S. Metref, E. Cosme, and R. Fablet (Nov. 2019). "Learning Generalized Quasi-Geostrophic Models Using Deep Neural Numerical Models". In: *arXiv:1911.08856 [physics, stat]*. arXiv: 1911.08856.

Li, D., K. Xu, J. M. Harris, and E. Darve (2020a). "Coupled time-lapse full-waveform inversion for subsurface flow problems using intrusive automatic differentiation". In: *Water Resources Research* 56.8, e2019WR027032.

Li, Z., N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar (2020b). "Fourier Neural Operator for Parametric Partial Differential Equations". In: *arXiv*. DOI: 10.48550/arxiv.2010.08895.

Lions, J. L. (1971). *Optimal control of systems governed by partial differential equations*. Vol. 170. Springer.

Logan, L. C., S. H. K. Narayanan, R. Greve, and P. Heimbach (2020). "SICOPOLIS-AD v1: an open-source adjoint modeling framework for ice sheet simulation enabled by the algorithmic differentiation tool OpenAD". In: *Geoscientific Model Development* 13.4, pp. 1845–1864.

Lyness, J. N. (1967). "Numerical algorithms based on the theory of complex variable". In: *Proceedings of the 1967 22nd national conference on -*, pp. 125–133. DOI: 10.1145/800196.805983.

Lyness, J. N. and C. B. Moler (1967). "Numerical Differentiation of Analytic Functions". In: *SIAM Journal on Numerical Analysis* 4.2, pp. 202–210. DOI: 10.1137/0704019.

Ma, Y., V. Dixit, M. J. Innes, X. Guo, and C. Rackauckas (2021a). "A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions". In: *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, pp. 1–9.

Ma, Y., V. Dixit, M. Innes, X. Guo, and C. Rackauckas (July 2021b). "A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions". In: *arXiv:1812.01892 [cs]*. arXiv: 1812.01892.

MacAyeal, D. R. (1992). "The basal stress distribution of Ice Stream E, Antarctica, inferred by control methods". In: *Journal of Geophysical Research: Solid Earth* 97.B1, pp. 595–603.

— (1993). "A tutorial on the use of control methods in ice-sheet modeling". In: *Journal of Glaciology* 39.131. Publisher: Cambridge University Press, pp. 91–98. DOI: 10.3189/S0022143000015744.

Magruder, L., T. Neumann, and N. Kurtz (2021). "ICESat-2 Early Mission Synopsis and Observatory Performance". In: *Earth and Space Science* 8.5, e2020EA001555.

Maguire, E., L. Heinrich, and G. Watt (Oct. 2017). "HEPData: a repository for high energy physics data". In: *Journal of Physics: Conference Series* 898.10, p. 102006. DOI: 10.1088/1742-6596/898/10/102006.

Manzyuk, O., B. A. Pearlmutter, A. A. Radul, D. R. Rush, and J. M. Siskind (2019). "Perturbation confusion in forward automatic differentiation of higher-order functions". In: *Journal of Functional Programming* 29, e12.

Mardia, K. V. (1975). "Distribution Theory for the Von Mises-Fisher Distribution and Its Application". In: *A Modern Course on Statistical Distributions in Scientific Work*. Ed. by G. P. Patil, S. Kotz, and J. K. Ord. Dordrecht: Springer Netherlands, pp. 113–130.

Mardia, K. V. and P. E. Jupp (2000). *Directional statistics*. Vol. 2. Wiley Online Library.

Margossian, C. C. (2018). "A Review of automatic differentiation and its efficient implementation". In: *arXiv*. DOI: 10.48550/arxiv.1811.05031.

Martins, J. R. R. A., P. Sturdza, and J. J. Alonso (2003). "The complex-step derivative approximation". In: *ACM Transactions on Mathematical Software (TOMS)* 29, pp. 245–262. DOI: 10.1145/838250.838251.

Martins, J., P. Sturdza, and J. Alonso (2001). "The connection between the complex-step derivative approximation and algorithmic differentiation". In: *39th Aerospace Sciences Meeting and Exhibit*, p. 921.

Marzio, M. D., A. Panzera, and C. C. Taylor (2019). "Nonparametric Rotations for Sphere-Sphere Regression". In: *Journal of the American Statistical Association* 114.525, pp. 466–476. DOI: 10.1080/01621459.2017.1421542.

Mathur, R. (2012). "An analytical approach to computing step sizes for finite-difference derivatives". PhD thesis.

Mattheakis, M., D. Sondak, A. S. Dogra, and P. Protopapas (2020). "Hamiltonian Neural Networks for solving differential equations". In: *arXiv*.

Maussion, F., A. Butenko, N. Champollion, M. Dusch, J. Eis, K. Fourteau, P. Gregor, A. H. Jarosch, J. Landmann, F. Oesterle, B. Recinos, T. Rothenpieler, A. Vlug, C. T. Wild, and B. Marzeion (Mar. 2019). "The Open Global Glacier Model (OGGM) v1.1". en. In: *Geoscientific Model Development* 12.3, pp. 909–931. DOI: 10.5194/gmd-12-909-2019.

Maussion, F., T. Rothenpieler, M. Dusch, P. Schmitt, A. Vlug, L. Schuster, N. Champollion, F. Li, B. Marzeion, M. Oberrauch, J. Eis, J. Landmann, A. Jarosch, A. Fischer, luzpaz, S. Hanus, D. Rounce, M. Castellani, S. L. Bartholomew, S. Minallah, bowenbelongstonature, C. Merrill, D. Otto, D. Loibl, L. Ultee, S. Thompson, anton-ub, P. Gregor, and zhaohongyu (Mar. 2023). *OGGM/oggm: v1.6.0*. Version v1.6.0. DOI: 10.5281/zenodo.7718476.

McElhinny, M. and P. McFadden (2000). *Paleomagnetism: Continents and Oceans*. Vol. 73. International Geophysics Series. San Diego: Academic Press.

McFadden, P., R. Merrill, and M. McElhinny (1988). "Dipole/quadrupole family modeling of paleosecular variation". In: *Journal of Geophysical Research: Solid Earth* 93.B10, pp. 11583–11588.

McFadden, P., R. Merrill, M. McElhinny, and S. Lee (1991). "Reversals of the Earth's magnetic field and temporal variations of the dynamo families". In: *Journal of Geophysical Research: Solid Earth* 96.B3, pp. 3923–3933.

McGreivy, N., S. Hudson, and C. Zhu (2021). "Optimized finite-build stellarator coils using automatic differentiation". In: *Nuclear Fusion* 61, p. 026020. DOI: 10.1088/1741-4326/abcd76.

McKinney, W. (2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

Mesnard, O. and L. A. Barba (2020). "Reproducible Workflow on a Public Cloud for Computational Fluid Dynamics". In: *Computing in Science Engineering* 22.1, pp. 102–116. DOI: 10.1109/mcse.2019.2941702.

Millan, R., J. Mouginot, A. Rabatel, and M. Morlighem (Feb. 2022). "Ice velocity and thickness of the world's glaciers". en. In: *Nature Geoscience* 15.2, pp. 124–129. DOI: 10.1038/s41561-021-00885-z.

Mitusch, S. K., S. W. Funke, and J. S. Dokken (2019). "dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake". In: *Journal of Open Source Software* 4.38, p. 1292. DOI: 10.21105/joss.01292.

Mogensen, P. K. and A. N. Riseth (2018). "Optim: A mathematical optimization package for Julia". In: *Journal of Open Source Software* 3.24, p. 615. DOI: 10.21105/joss.00615.

Mohajerani, Y., M. Wood, I. Velicogna, and E. Rignot (Jan. 2019). "Detection of Glacier Calving Margins with Convolutional Neural Networks: A Case Study". en. In: *Remote Sensing* 11.1, p. 74. DOI: 10.3390/rs11010074.

Mohammadi, B. and O. Pironneau (2004). "Shape optimization in fluid mechanics". In: *Annual Review of Fluid Mechanics* 36.1, pp. 255–279. DOI: 10.1146/annurev.fluid.36.050802.121926.

— (2009). *Applied shape optimization for fluids*. OUP Oxford.

Monnier, J. and P.-E. d. Boscs (2017). "Inference of the bottom properties in shallow ice approximation models". In: *Inverse Problems* 33.11, p. 115001. DOI: 10.1088/1361-6420/aa7b92.

Morlighem, M., E. Rignot, T. Binder, D. Blankenship, R. Drews, G. Eagles, O. Eisen, F. Ferraccioli, R. Forsberg, P. Fretwell, et al. (2020). "Deep glacial troughs and stabilizing ridges unveiled beneath the margins of the Antarctic ice sheet". In: *Nature geoscience* 13.2, pp. 132–137.

Moses, W. and V. Churavy (2020). "Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 12472–12485.

Moses, W. S., V. Churavy, L. Paehler, J. Hückelheim, S. H. K. Narayanan, M. Schanen, and J. Doerfert (2021). "Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme". In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–16.

Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

Nanni, U., D. Scherler, F. Ayoub, R. Millan, F. Herman, and J.-P. Avouac (2023). "Climatic control on seasonal variations in mountain glacier surface velocity". In: *The Cryosphere* 17.4, pp. 1567–1583. DOI: 10.5194/tc-17-1567-2023.

National Academies of Sciences and Division on Engineering and Physical Sciences, Space Studies Board, and Committee on Best Practices for a Future Open Code Policy for NASA Space Science (2018). "Open Source Software Policy Options for NASA Earth and Space Sciences". In.

Naumann, U. (2011). *The Art of Differentiating Computer Programs*. Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9781611972078.

Neuenhofen, M. (2018). "Review of theory and implementation of hyper-dual numbers for first and second order automatic differentiation". In: *arXiv*. DOI: 10.48550/arxiv.1801.03614.

Norcliffe, A. and M. P. Deisenroth (2023). "Faster Training of Neural ODEs Using Gauß-Legendre Quadrature". In: *arXiv*. DOI: 10.48550/arxiv.2308.10644.

Oden, J. T., R. Moser, and O. Ghattas (2010). "Computer Predictions with Quantified Uncertainty, Part II". In: *SIAM News* 43.10, pp. 1–4.

Oktay, D., N. McGreivy, J. Aduol, A. Beatson, and R. P. Adams (2020). "Randomized Automatic Differentiation". In: *arXiv*. DOI: 10.48550/arxiv.2007.10412.

Onken, D. and L. Ruthotto (2020). "Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows". In: *arXiv*. DOI: 10.48550/arxiv.2005.13420.

Opdyke, M. D. and J. E. Channell (1996). *Magnetic stratigraphy*. Academic Press.

Palmieri, G., M. Tiboni, and G. Legnani (2020). "Analysis of the Upper Limitation of the Most Convenient Cadence Range in Cycling Using an Equivalent Moment Based Cost Function". In: *Mathematics* 8.11. DOI: 10.3390/math8111947.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8026–8037.

Pathak, J., S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, P. Hassanzadeh, K. Kashinath, and A. Anandkumar (2022). "FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators". In: *arXiv*. DOI: 10.48550/arxiv.2202.11214.

Peckham, S. D. (2014). "The CSDMS standard names: Cross-domain naming conventions for describing process models, data sets and their associated variables". In.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12, pp. 2825–2830.

Pérez, F. and B. E. Granger (2007). "IPython: A System for Interactive Scientific Computing". In: *Computing in Science Engineering* 9.3, pp. 21–29. DOI: 10.1109/MCSE.2007.53.

Perkel, J. M. (2021). "Ten computer codes that transformed science". In: *Nature* 589.7842, pp. 344–348. DOI: 10.1038/d41586-021-00075-2.

Pestourie, R., Y. Mroueh, C. Rackauckas, P. Das, and S. G. Johnson (2023). "Physics-enhanced deep surrogates for partial differential equations". In: *Nature Machine Intelligence* 5.12, pp. 1458–1465. DOI: 10.1038/s42256-023-00761-y.

Pförtner, M., I. Steinwart, P. Hennig, and J. Wenger (2022). "Physics-Informed Gaussian Process Regression Generalizes Linear PDE Solvers". In: *arXiv*. DOI: 10.48550/arxiv.2212.12474.

Pironneau, O. (2005). "Optimal shape design for elliptic systems". In: *System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981*. Springer, pp. 42–66.

Pörtner, H.-O., D. Roberts, H. Adams, I. Adelekan, C. Adler, R. Adrian, P. Aldunce, E. Ali, R. A. Begum, B. B. Friedl, R. B. Kerr, R. Biesbroek, J. Birkmann, K. Bowen, M. Caretta, J. Carnicer, E. Castellanos, T. Cheong, W. Chow, G. C. G. Cissé, and Z. Z. Ibrahim (2022). *Climate Change 2022: Impacts, Adaptation and Vulnerability*. Technical Summary. Cambridge, UK and New York, USA: Cambridge University Press, pp. 37–118.

Rackauckas, C., M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit (2019). "DiffEqFlux.jl - A Julia Library for Neural Differential Equations". In: *arXiv*.

Rackauckas, C., A. Edelman, K. Fischer, M. Innes, E. Saba, V. B. Shah, and W. Tebbutt (2021). "Generalized physics-informed learning through language-wide differentiable programming". In.

Rackauckas, C., Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman (2020). "Universal differential equations for scientific machine learning". In: *arXiv preprint arXiv:2001.04385*.

Rackauckas, C. and Q. Nie (2016). "DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia". In: *Journal of Open Research Software* 5.1, p. 15. DOI: 10.5334/jors.151.

Raissi, M., P. Perdikaris, and G. Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

Ramadhan, A., J. C. Marshall, A. N. Souza, X. K. Lee, U. Piterbarg, A. Hillier, G. L. Wagner, C. Rackauckas, C. Hill, J.-M. Campin, and R. Ferrari (2022). "Capturing missing physics in climate model parameterizations using neural differential equations". In: DOI: 10.1002/essoar.10512533.1.

Ramsay, J. and G. Hooker (2017). *Dynamic Data Analysis, Modeling Data with Differential Equations*. DOI: 10.1007/978-1-4939-7190-9.

Ramsundar, B., D. Krishnamurthy, and V. Viswanathan (2021). "Differentiable Physics: A Position Piece". In: *arXiv*. DOI: 10.48550/arxiv.2109.07573.

Ranocha, H., L. Dalcin, M. Parsani, and D. I. Ketcheson (2022). “Optimized Runge-Kutta Methods with Automatic Step Size Control for Compressible Computational Fluid Dynamics”. In: *Communications on Applied Mathematics and Computation* 4.4. Paper with the RDPK3Sp35 method, pp. 1191–1228. DOI: 10.1007/s42967-021-00159-w.

Rasp, S., M. S. Pritchard, and P. Gentine (Sept. 2018). “Deep learning to represent subgrid processes in climate models”. en. In: *Proceedings of the National Academy of Sciences* 115.39, pp. 9684–9689. DOI: 10.1073/pnas.1810286115.

Razavi, S., A. Jakeman, A. Saltelli, C. Prieur, B. Iooss, E. Borgonovo, E. Plischke, S. L. Piano, T. Iwanaga, W. Becker, S. Tarantola, J. H. A. Guillaume, J. Jakeman, H. Gupta, N. Melillo, G. Rabitti, V. Chabridon, Q. Duan, X. Sun, S. Smith, R. Sheikholeslami, N. Hosseini, M. Asadzadeh, A. Puy, S. Kucherenko, and H. R. Maier (2021). “The Future of Sensitivity Analysis: An essential discipline for systems modeling and policy support”. In: *Environmental Modelling & Software* 137, p. 104954. DOI: 10.1016/j.envsoft.2020.104954.

Revels, J., M. Lubin, and T. Papamarkou (2016). “Forward-Mode Automatic Differentiation in Julia”. In: *arXiv:1607.07892 [cs.MS]*.

RGI 7.0 Consortium, 2. R. G. I. (2023). *Randolph Glacier Inventory 7.0*. English. DOI: 10.5067/f6jmovy5navz.

Riel, B., B. Minchew, and T. Bischoff (2021). “Data-Driven Inference of the Mechanics of Slip Along Glacier Beds Using Physics-Informed Neural Networks: Case Study on Rutford Ice Stream, Antarctica”. In: *Journal of Advances in Modeling Earth Systems* 13.11. e2021MS002621 2021MS002621, e2021MS002621. DOI: https://doi.org/10.1029/2021MS002621.

Roberts, D. A. (2021). “Why is AI hard and Physics simple?” In: *arXiv*.

Rolnick, D., P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown, A. S. Luccioni, T. Maharaj, E. D. Sherwin, S. K. Mukkavilli, K. P. Kording, C. P. Gomes, A. Y. Ng, D. Hassabis, J. C. Platt, F. Creutzig, J. Chayes, and Y. Bengio (2022). “Tackling Climate Change with Machine Learning”. In: *ACM Computing Surveys (CSUR)* 55.2, pp. 1–96. DOI: 10.1145/3485128.

Rosenthal, M., W. Wu, E. Klassen, and A. Srivastava (2014). “Spherical Regression Models Using Projective Linear Transformations”. In: *Journal of the American Statistical Association* 109.508, pp. 1615–1624. DOI: 10.1080/01621459.2014.892881.

Rüde, U., K. Willcox, L. C. McInnes, and H. D. Sterck (2018). “Research and Education in Computational Science and Engineering”. In: *SIAM Review* 60.3, pp. 707–754. DOI: 10.1137/16m1096840.

Ruder, S. (2016). “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747*.

Runge, J., S. Bathiany, E. Bollt, G. Camps-Valls, D. Coumou, E. Deyle, C. Glymour, M. Kretschmer, M. D. Mahecha, J. Muñoz-Marí, E. H. v. Nes, J. Peters, R. Quax, M. Reichstein, M. Scheffer, B. Schölkopf, P. Spirtes, G. Sugihara, J. Sun, K. Zhang, and J.

Zscheischler (2019). “Inferring causation from time series in Earth system sciences”. In: *Nature Communications* 10.1, p. 2553. DOI: 10.1038/s41467-019-10105-3.

Samir, C., P.-A. Absil, A. Srivastava, and E. Klassen (2012). “A Gradient-Descent Method for Curve Fitting on Riemannian Manifolds”. In: *Foundations of Computational Mathematics* 12.1, pp. 49–73. DOI: 10.1007/s10208-011-9091-7.

Sandu, A. (2006). “On the properties of Runge-Kutta discrete adjoints”. In: *Computational Science–ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part IV 6*. Springer, pp. 550–557.

— (2011). “Solution of inverse problems using discrete ODE adjoints”. In: *Large-Scale Inverse Problems and Quantification of Uncertainty*, pp. 345–365.

Sapienza, F. et al. (2024a). “An Analytical Model of Magnetic Field Draping in Induced Magnetospheres”. In: *preparation*.

— (2024b). “Fitting curves in the sphere using universal differential equations”. In: *preparation*.

Sapienza, F., J. Bolibar, F. Schäfer, P. Heimbach, G. Hooker, F. Pérez, P. Persson, C. Rackauckas, V. Boussange, B. Groenke, and A. Pal (2024c). “Differentiable Programming for Differential Equations: A Review”. In: *preparation*.

Sapienza, F., L. C. Gallo, Y. Zhang, B. Vaes, M. Domeier, and N. L. Swanson-Hysell (2023a). “Quantitative Analysis of Paleomagnetic Sampling Strategies”. In: *Journal of Geophysical Research: Solid Earth* 128.11, e2023JB027211. DOI: https://doi.org/10.1029/2023JB027211.

Sapienza, F., L. C. Gallo, Y. Zhang, B. Vaes, M. Domeier, and N. Swanson-Hysell (2023b). *PolarWandering/PaleoSampling (Version 1.0.0)*. Comp. software. Version 1.0.0. DOI: https://doi.org/10.5281/zenodo.8347149.

Schäfer, F., M. Tarek, L. White, and C. Rackauckas (2021). “AbstractDifferentiation.jl: Backend-Agnostic Differentiable Programming in Julia”. In: *arXiv*. DOI: 10.48550/arxiv.2109.12449.

Schanen, M., S. H. K. Narayanan, S. Williamson, V. Churavy, W. S. Moses, and L. Paehler (2023). “Transparent Checkpointing for Automatic Differentiation of Program Loops Through Expression Transformations”. In: ed. by J. Mikyška, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. Sloot, pp. 483–497.

Schleder, G. R., A. C. M. Padilha, C. M. Acosta, M. Costa, and A. Fazzio (2019). “From DFT to machine learning: recent approaches to materials science–a review”. In: *Journal of Physics: Materials* 2.3, p. 032001. DOI: 10.1088/2515-7639/ab084b.

Schneider, T., J. Teixeira, C. S. Bretherton, F. Brient, K. G. Pressel, C. Schär, and A. P. Siebesma (2017). “Climate goals and computing the future of clouds”. In: *Nature Climate Change* 7.1, pp. 3–5.

Schoof, C. (2006). “A variational approach to ice stream flow”. In: *Journal of Fluid Mechanics* 556, pp. 227–251. DOI: 10.1017/s0022112006009591.

Schroeder, D. M. (2022). “Paths forward in radioglaciology”. In: *Annals of Glaciology* 63.87–89, pp. 13–17. DOI: 10.1017/aog.2023.3.

Serban, R. and A. C. Hindmarsh (2005). “CVODES: the sensitivity-enabled ODE solver in SUNDIALS”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 47438, pp. 257–269.

Shen, C., A. P. Appling, P. Gentine, T. Bandai, H. Gupta, A. Tartakovsky, M. Baity-Jesi, F. Fenicia, D. Kifer, L. Li, X. Liu, W. Ren, Y. Zheng, C. J. Harman, M. Clark, M. Farthing, D. Feng, P. Kumar, D. Aboelyazeed, F. Rahmani, Y. Song, H. E. Beck, T. Bindas, D. Dwivedi, K. Fang, M. Höge, C. Rackauckas, B. Mohanty, T. Roy, C. Xu, and K. Lawson (2023). “Differentiable modelling to unify machine learning and physical models for geosciences”. In: *Nature Reviews Earth & Environment*, pp. 1–16. DOI: 10.1038/s43017-023-00450-9.

Silva, H. D., J. L. Gustafson, and W.-F. Wong (2018). “Making Strassen Matrix Multiplication Safe”. In: *2018 IEEE 25th International Conference on High Performance Computing (HiPC)* 00, pp. 173–182. DOI: 10.1109/hipc.2018.00028.

Sirignano, J. and K. Spiliopoulos (2017). “DGM: A deep learning algorithm for solving partial differential equations”. In: *arXiv*. DOI: 10.1016/j.jcp.2018.08.029.

Sirkes, Z. and E. Tziperman (1997). “Finite Difference of Adjoint or Adjoint of Finite Difference?” In: *Monthly Weather Review* 125.12, pp. 3373–3378. DOI: 10.1175/1520-0493(1997)125<3373:fdoaoa>2.0.co;2.

Siskind, J. M. and B. A. Pearlmutter (2005). “Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD”. In.

Smith, J. S., O. Isayev, and A. E. Roitberg (2017). “ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost”. In: *Chemical science* 8.4, pp. 3192–3203.

Smucler, E., F. Sapienza, and A. Rotnitzky (2022). “Efficient adjustment sets in causal graphical models with hidden variables”. In: *Biometrika* 109.1, pp. 49–65.

Snow, T., J. Millstein, J. Scheick, W. Sauthoff, W. J. Leong, J. Colliander, F. Pérez, J. Munroe, D. Felikson, T. Sutterley, and M. Siegfried (2023). *CryoCloud JupyterBook*. DOI: https://doi.org/10.5281/zenodo.7576602.

Spain, B. (2003). *Tensor Calculus: a concise course*. Courier Corporation.

Squire, W. and G. Trapp (1998). “Using Complex Variables to Estimate Derivatives of Real Functions”. In: 40, pp. 110–112. DOI: 10.1137/s003614459631241x.

Stein, E. M. and R. Shakarchi (2010). *Complex analysis*. Vol. 2. Princeton University Press.

Stoer, J. and R. Bulirsch (2002). *Introduction to numerical analysis*. Springer.

Stoudt, S., V. N. Vásquez, and C. C. Martinez (2021). “Principles for data analysis workflows”. In: *PLOS Computational Biology* 17.3, e1008770. DOI: 10.1371/journal.pcbi.1008770.

Strouwen, A., B. M. Nicolaï, and P. Goos (Jan. 12, 2022). “Robust Dynamic Experiments for the Precise Estimation of Respiration and Fermentation Parameters of Fruit and Vegetables”. In: *PLOS Computational Biology* 18.1. Ed. by P. Mendes, e1009610. DOI: 10.1371/journal.pcbi.1009610.

Synge, J. L. and A. Schild (1978). *Tensor calculus*. Vol. 5. Courier Corporation.

Tauxe, L., R. Shaar, L. Jonestrask, N. Swanson-Hysell, R. Minnett, A. Koppers, C. Constable, N. Jarboe, K. Gaastra, and L. Fairchild (2016). “PmagPy: Software package for paleomagnetic data analysis and a bridge to the Magnetics Information Consortium (MagIC) Database”. In: *Geochemistry, Geophysics, Geosystems* 17.6, pp. 2450–2463.

Tauxe, L. (2010). *Essentials of Paleomagnetism*. University of California Press.

Tauxe, L., C. Constable, C. L. Johnson, A. A. Koppers, W. R. Miller, and H. Staudigel (2003). “Paleomagnetism of the southwestern USA recorded by 0–5 Ma igneous rocks”. In: *Geochemistry, Geophysics, Geosystems* 4.4.

Tauxe, L. and D. V. Kent (2004). “Timescales Of The Paleomagnetic Field”. In: *Geophysical Monograph Series*, pp. 101–115. DOI: 10.1029/145gm08.

Teisberg, T. O., D. M. Schroeder, and E. J. MacKie (2021). “A Machine Learning Approach to Mass-Conserving Ice Thickness Interpolation”. In: *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS* 00, pp. 8664–8667. DOI: 10.1109/igarss47720.2021.9555002.

The icepyx Developers (2023). *icepyx: Python tools for obtaining and working with ICESat-2 data*. icesat2py. DOI: https://doi.org/10.5281/zenodo.7729175.

The World Economic Forum (2018). “Harnessing artificial intelligence for the earth”. In: *Tech. Rep.*

Thuerey, N., P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um (2021). *Physics-based Deep Learning*. WWW.

Tibshirani, R. J. (2014). “Adaptive piecewise polynomial estimation via trend filtering”. In: *The Annals of Statistics* 42.1, pp. 285–323. DOI: 10.1214/13-AOS1189.

Torsvik, T. H., R. Van der Voo, U. Preeden, C. Mac Niocaill, B. Steinberger, P. V. Doubrovine, D. J. Van Hinsbergen, M. Domeier, C. Gaina, E. Tohver, et al. (2012). “Phanerozoic polar wander, palaeogeography and dynamics”. In: *Earth-Science Reviews* 114.3-4, pp. 325–368.

Tsitouras, C. (2011). “Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption”. In: *Computers & Mathematics with Applications* 62.2, pp. 770–775. DOI: 10.1016/j.camwa.2011.06.002.

Turk, M. J. (2013). “How to Scale a Code in the Human Dimension”. In: *arXiv*.

Vaes, B., S. Li, C. G. Langereis, and D. J. van Hinsbergen (2021). “Reliability of palaeomagnetic poles from sedimentary rocks”. In: *Geophysical Journal International* 225.2, pp. 1281–1303.

Vandamme, D. (1994). “A new method to determine paleosecular variation”. In: *Physics of the Earth and Planetary Interiors* 85.1-2, pp. 131–142.

Walther, A. (2007). “Automatic differentiation of explicit Runge-Kutta methods for optimal control”. In: *Computational Optimization and Applications* 36.1, pp. 83–108. DOI: 10.1007/s10589-006-0397-3.

Wang, F., D. Zheng, J. Decker, X. Wu, G. M. Essertel, and T. Rompf (2019). “Backpropagation with Continuation Callbacks:Foundations for Efficient and ExpressiveDifferentiable Programming”. In: *Proceedings of the ACM on Programming Languages* 3.ICFP, p. 96. DOI: 10.1145/3341700.

Wang, Q., R. Hu, and P. Blonigan (2014). "Least Squares Shadowing sensitivity analysis of chaotic limit cycle oscillations". In: *Journal of Computational Physics* 267, pp. 210–224. DOI: https://doi.org/10.1016/j.jcp.2014.03.002.

Wang, Y., C.-Y. Lai, J. Gómez-Serrano, and T. Buckmaster (2023). "Asymptotic Self-Similar Blow-Up Profile for Three-Dimensional Axisymmetric Euler Equations Using Neural Networks". In: *Physical Review Letters* 130.24, p. 244002. DOI: 10.1103/physrevlett.130.244002.

Wang, Y., C.-Y. Lai, and C. Cowen-Breen (2022). "Discovering the rheology of Antarctic Ice Shelves via physics-informed deep learning". In: DOI: 10.21203/rs.3.rs-2135795/v1.

Wanner, G. and E. Hairer (1996). *Solving ordinary differential equations II*. Vol. 375. Springer Berlin Heidelberg New York.

Watson, G. S. (1982). "Distributions on the circle and sphere". In: *Journal of Applied Probability* 19.A, pp. 265–280. DOI: 10.1017/s0021900200034628.

Watt-Meyer, O., G. Dresdner, J. McGibbon, S. K. Clark, B. Henn, J. Duncan, N. D. Brenowitz, K. Kashinath, M. S. Pritchard, B. Bonev, M. E. Peters, and C. S. Bretherton (2023). "ACE: A fast, skillful learned global atmospheric model for climate prediction". In: *arXiv*. DOI: 10.48550/arxiv.2310.02074.

Wengert, R. E. (1964). "A simple automatic derivative evaluation program". In: *Communications of the ACM* 7.8, pp. 463–464. DOI: 10.1145/355586.364791.

Weyand, T., I. Kostrikov, and J. Philbin (2016). "Planet-photo geolocation with convolutional neural networks". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer, pp. 37–55.

Whillans, I. M. (1977). "The Equation of Continuity and its Application to the Ice Sheet Near "byrd" Station, Antarctica". In: *Journal of Glaciology* 18.80, pp. 359–371. DOI: 10.3189/S0022143000021055.

Wigner, E. P. (1960). "The unreasonable effectiveness of mathematics in the natural sciences". In: *Communications on Pure and Applied Mathematics* 13, pp. 1–14. DOI: 10.1002/cpa.3160130102.

Wolfe, P. (1982). "Checking the Calculation of Gradients". In: *ACM Transactions on Mathematical Software (TOMS)* 8.4, pp. 337–343. DOI: 10.1145/356012.356013.

Wu, H., S. Y. Greer, and D. O'Malley (2023). "Physics-embedded inverse analysis with algorithmic differentiation for the earth's subsurface". In: *Scientific Reports* 13.1, p. 718. DOI: 10.1038/s41598-022-26898-1.

Zdeborová, L. (May 2020). "Understanding deep learning is also a job for physicists". en. In: *Nature Physics*. DOI: 10.1038/s41567-020-0929-2.

Zekollari, H., M. Huss, and D. Farinotti (Apr. 2019). "Modelling the future evolution of glaciers in the European Alps under the EURO-CORDEX RCM ensemble". en. In: *The Cryosphere* 13.4, pp. 1125–1146. DOI: 10.5194/tc-13-1125-2019.

Zhang, H. and A. Sandu (2014). "FATODE: A library for forward, adjoint, and tangent linear integration of ODEs". In: *SIAM Journal on Scientific Computing* 36.5, pp. C504–C523.

Zhao, C., R. M. Gladstone, R. C. Warner, M. A. King, T. Zwinger, and M. Morlighem (Aug. 2018). “Basal friction of Fleming Glacier, Antarctica – Part 1: Sensitivity of inversion to temperature and bedrock uncertainty”. en. In: *The Cryosphere* 12.8, pp. 2637–2652. DOI: 10.5194/tc-12-2637-2018.

Zhou, T., X. Wan, D. Z. Huang, Z. Li, Z. Peng, A. Anandkumar, J. F. Brady, P. W. Sternberg, and C. Daraio (2024). “AI-aided geometric design of anti-infection catheters”. In: *Science Advances* 10.1, eadj1741. DOI: 10.1126/sciadv.adj1741.

Zhu, Q. and J. Yang (2021a). “A Local Deep Learning Method for Solving High Order Partial Differential Equations”. In: *arXiv*. DOI: 10.48550/arxiv.2103.08915.

Zhu, W., K. Xu, E. Darve, and G. C. Beroza (2021b). “A general approach to seismic inversion with automatic differentiation”. In: *Computers & Geosciences* 151, p. 104751. DOI: 10.1016/j.cageo.2021.104751.

Zhuang, J., N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan (2020). “Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE.” In: *Proceedings of machine learning research* 119, pp. 11639–11649.

Zubov, K., Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe, N. Vinchhi, K. Balakrishnan, D. Upadhyay, and C. Rackauckas (2021). “NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations”. In: *arXiv*. DOI: 10.48550/arxiv.2107.09443.

Appendix A

Supplementary code

This is a list of the code provided along with the current manuscript. All the following scripts can be found in the GitHub repository `DiffEqSensitivity-Review`.

♣1 **Comparison of direct methods.** The script `direct-comparision.jl` reproduces Figure 2.6.

♣2 **Dual numbers definition.** The script `dualnumber_definition.jl` includes a very simple example of how to define a dual number using `struct` in Julia and how to extend simple unary and binary operations to implement the chain rule usign multiple distpatch.

♣3 **When AD is algorithmically correct but numerically wrong.** The script `example-AD-tolerances.jl` includes the example shown in Section 2.4.1.2.1 where forward AD gives the wrong answer when tolerances in the gradient are not computed taking into account both numerical errors in the numerical solution and the sensitivity matrix. Further examples of this phenomena can be found in the Python script `testgradient_python.py` and the Julia `testgradient_julia.jl`.

♣4 **Complex step in numerical solver.** The script `complex_solver.jl` shows how to define the dynamics of the ODE to support complex variables and then compute the complex step derivative.

♣5 **Solving the sensitivity equation.** The scrip `sensitivityequation.jl` includes a manual implementation of the sensitivity equations. This also includes how to compute the same sensitivity using `ForwardSensitivity` in Julia.

Appendix B

Shallow shelf approximation

The SIA approximation suits well in cases where we can think of the glacier as a thin layer of ice and then ignore the effect of longitudinal and lateral stresses. However, *"A peculiar behaviour exhibited by ice sheets, which sets them apart from other geophysical thin-film flows, is the formation of ice streams. These bands of fast-flowing ice within an ice sheet are typically around fifty kilometres wide and hundreds of kilometres long, and are surrounded by more slowly moving ice often termed ice ridges. The high flow velocities of ice streams generally cannot be explained by vertical shearing in the ice, as one would expect from a typical lubrication flow, but must be caused by rapid sliding at the contact between ice and the underlying bed"* (Schoof 2006).

In these cases, it is important to incorporate the effect of lateral and longitudinal stresses that balance the driving stress in addition to the usual basal drag (this are components τ_{xx}, τ_{yy} and τ_{xy} of the deviatoric stress). The simplest form of a membrane stress balance we can derive from the Stokes model is SSA (Bueler et al. 2009).

In opposition to the SIA equation, the SSA model uses Glen's Law in its viscous form (Section B.1). Since the driving stress is not longer balanced just by the basal drag, we need an expression for the friction with the bed (Section B.2), which will correspond to the Mohn-Coulomb law. With all this ingredients, we will be ready to derive the SSA equations.

B.1 Glen's law in viscosity form

The SSA equation uses Glen's Law too but in its viscosity form. Instead of writing the strain rate $\dot{\epsilon}_{ij}$ as a function of the deviatoric stress tensor, we do the other way around. Glen's law states

$$\dot{\epsilon}_{ij} = A\tau_E^{n-1}\tau_{ij}, \tag{B.1}$$

with $2\tau_E^2 = \tau_{ij}\tau_{ij}$. Using Glen's law again in the expression for τ_E^2 we obtain

$$2\tau_E^2 = \frac{1}{A^2\tau_E^{2n-2}}\dot{\epsilon}_{ij}\dot{\epsilon}_{ij} \quad\Rightarrow\quad \tau_E^{2n} = \frac{1}{2A^2}\dot{\epsilon}_{ij}\dot{\epsilon}_{ij} = \frac{1}{A^2}\dot{\epsilon}_E^2 \quad\Rightarrow\quad \tau_E^{n-1} = A^{-(n-1)/n}\dot{\epsilon}_E^{(n-1)/n} \tag{B.2}$$

and then

$$\dot{\epsilon}_{ij} = A^{1/n}\dot{\epsilon}_E^{1-\frac{1}{n}}\tau_{ij} \tag{B.3}$$

and finally

$$\tau_{ij} = 2\nu(\dot{\epsilon}_E)\dot{\epsilon}_{ij}, \tag{B.4}$$

where ν is the non-linear viscosity satisfying $2\nu = B\dot{\epsilon}_E^{\frac{1}{n}-1}$ and $B = A^{-1/n}$ the ice hardness.

B.2 Friction with the bed

The form of friction law depends on the debris concentration (rocks) and weather the ice contacts the bed directly or transmit is weight thought the particles. Coulomb friction acts between the debris particles and the bed, giving a total force per unit area $\mathcal{A}$ (drag force)

$$\tau_f = \mu P_c \mathcal{A}_c \,/\, \mathcal{A}, \tag{B.5}$$

with $\mu \approx 0.6$ the friction coefficient, P_c the pressure at regions of contact and $\mathcal{A}_c$ the area of contact.

The normal stress at the bed must balance the weight of the ice column $P_i = \rho_i g H$ and then

$$P_i \mathcal{A} = P_w \mathcal{A}_w + P_c \mathcal{A}_c,$$

with P_w the water pressure applied on some area $\mathcal{A}_w$. Assuming $\mathcal{A} = \mathcal{A}_c + \mathcal{A}_w$ we obtain

$$\tau_f = \mu(P_i - f_w P_w), \tag{B.6}$$

with $f_w = \mathcal{A}_w \,/\, \mathcal{A}$ the fraction of the bed surface that is wet. If $f_w \approx 1$, this formula reduces to the Mohr-Coulomb relation

$$\tau_f = \mu N = \mu(P_i - P_w), \tag{B.7}$$

where N is the effective pressure, (Cuffey et al. 2010).

Now, a similar relation holds for the case of a deforming till. A deforming till cannot tolerate a stress no larger than τ_*. Laboratory experiments show that τ_* also follows a Mohr-Coulomb law

$$\tau_* = c_0 + fN = c_0 + \tan\varphi(\rho_i g H - P_w), \tag{B.8}$$

with c_0 the apparent cohesion. We can write $f = \tan\varphi$, where φ is the angle of the bed deformation with respect to the bed slope (angle β in Figure 7.3 in (Cuffey et al. 2010)).

Either τ_f or τ_* represent the absolute value of the force by unit area applied on the bed as a consequence of the friction between ice and rock. That force is applied in the opposite direction of movement of the glacier at the bed. If $u = (u, v)$ are the horizontal components of the velocity vector in the bed, then the basal drag τ_b components are

$$\tau_{b,x} = -\tau_* \frac{u}{\|u\|} \qquad \tau_{b,y} = -\tau_* \frac{u}{\|u\|}, \tag{B.9}$$

with $\|u\| = \sqrt{u^2 + v^2}$.

B.3 SSA derivation

We are going to consider a coordinate system with z pointing vertically and both x and y are in the horizontal plane. Now, the shallow approximation consists in assuming that the ice thickness is small compared with the horizontal dimensions of the glacier. Then, if ϵ denotes the ratio between the typical thickness and the typical horizontal extent of the glacier, we are going to consider the following scaling of the variables

$$(x, y, z) \leftarrow (x, y, \epsilon z) \tag{B.10}$$

$$(\tau_{xx}, \tau_{yy}, \tau_{zz}, \tau_{xy}) \leftarrow (\tau_{xx}, \tau_{yy}, \tau_{zz}, \tau_{xy}) \tag{B.11}$$

$$(\tau_{xz}, \tau_{yz}) \leftarrow \epsilon(\tau_{xz}, \tau_{yz}) \tag{B.12}$$

and consider all the equation at the lowest order in ϵ.

If σ_{ij} denotes the stress tensor, then the set of balance equations in 3 dimensions is

$$\frac{\partial \sigma_{ij}}{\partial x_j} - \rho g \sigma_{i3} = 0, \tag{B.13}$$

where the repeated index j indicates sum over $j = 1, 2, 3$ (ie, $j = x, y, z$) and the free index i indicates that this equation holds for $i = 1, 2, 3$. We can write these last equations as a function of the deviatoric stress $\tau_{ij} = \sigma_{ij} - \sigma_M$, with $\sigma_M = \sigma_{ii}$. Since σ_M follows the same sign convention as the stresses, it is negative for compression (Cuffey et al. 2010). Here, we are going to use the pressure $P = -\sigma_M$. Using $\sigma_{ij} = \tau_{ij} - P\delta_{ij}$ we obtain

$$\frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial P}{\partial x_i} - \rho g \delta_{i3} = 0. \tag{Balance equations}$$

for $i = z$ this last equation is

$$\underbrace{\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y}}_{O(\epsilon)} + \underbrace{\frac{\partial (\tau_{zz} - P)}{\partial z}}_{O(\epsilon^{-1})} - \rho g = 0 \quad \rightsquigarrow \quad \frac{\partial(\tau_{zz} - P)}{\partial z} = 0 \tag{B.14}$$

where $\rightsquigarrow$ denotes the result after removing higher order terms on ϵ in the equation.

First, we need to specify the boundary conditions in both the bed and surface. Define n^b and n^s the normal vectors in the bed and surface, respectively, given by

$$n^b = \frac{1}{\sqrt{1 + (\partial b/\partial x)^2 + (\partial b/\partial y)^2}} \left(-\frac{\partial b}{\partial x}, -\frac{\partial b}{\partial y}, 1 \right), \tag{B.15}$$

$$n^s = \frac{1}{\sqrt{1 + (\partial s/\partial x)^2 + (\partial s/\partial y)^2}} \left(-\frac{\partial s}{\partial x}, -\frac{\partial s}{\partial y}, 1 \right). \tag{B.16}$$

On the normal direction in the surface, the net force has to be zero, that is,

$$\sigma_{ij}^s n_j = \tau_{ij}^s n_j^s - P n_i^s = 0. \tag{Boundary condition in surface}$$

For $i = 3$ we obtain

$$\tau_{zz}^s \underbrace{- \frac{\partial s}{\partial x}\tau_{xz}^s - \frac{\partial s}{\partial y}\tau_{yz}^s}_{O(\epsilon^2)} - P = 0 \quad \rightsquigarrow \quad \tau_{zz}^s = P. \tag{B.17}$$

Equations (B.14) and (B.17) imply $P = \tau_{zz} = -\tau_{xx} - \tau_{yy}$. Also, for $i = 1$ we have (using $P = \tau_{zz}^s$)

$$\tau_{xz}^s - \frac{\partial s}{\partial x}\tau_{xx}^s - \frac{\partial s}{\partial y}\tau_{xy}^s + \frac{\partial s}{\partial x}\tau_{zz}^s = 0. \tag{B.18}$$

Notice that all the terms in the last equation are $O(\epsilon)$ and that is the reason why we keep them all.

The boundary conditions for the bed are a little bit more tricky. The normal that the bed experiences is a stress (force per unit area) acting in the direction $-n^b$ and with intensity

$$N = P - \tau_{ij}^b n_i^b n_j^b = P + O(\epsilon). \tag{B.19}$$

The total force per unit area experienced on the bed is a combination of the driving stress and the normal force and all these is balanced by the basal drag τ_i^b:

$$\tau_i^b + \tau_{ij}^b n^b - \tau_{jk}^b n_j^b n_k^b n_i^b = 0. \qquad \text{(Boundary condition in bed)}$$

Now, the boundary condition on the bed for $i = 1$, and using the Mohn-Coulomb law,

$$\tau_i^b = -\mu(N - p_w)\frac{u_i}{\|u\|}, \tag{B.20}$$

is equivalent to

$$\underbrace{\tau_{xz}^b - \frac{\partial b}{\partial x}\tau_{xx}^b - \frac{\partial b}{\partial y}\tau_{xy}^b + \frac{\partial b}{\partial x}\tau_{zz}^b}_{O(\epsilon)} + O(\epsilon^2) = \mu\underbrace{(P - p_w)\frac{u_x}{\|u\|}}_{O(\epsilon)}. \tag{B.21}$$

The right hand side of last equation is $O(\epsilon)$ because in practice $P - p_w = (\epsilon)$ (they both increase linearly with depth).

We can now vertically integrate the balance equation for x (or y) and obtain

$$\int_b^s \frac{\partial \tau_{xx}}{\partial x}dz + \int_b^z \frac{\partial \tau_{xy}}{\partial y}dz + \tau_{xz}^s - \tau_{xz}^b - \int_b^s \frac{\partial P}{\partial x}dz = 0. \tag{B.22}$$

Using Leibniz integral rule we obtain

$$0 = \frac{\partial}{\partial x}\int_b^s \tau_{xx}dz \qquad + \frac{\partial b}{\partial x}\tau_{xx}^b \qquad - \frac{\partial s}{\partial x}\tau_{xx}^s \tag{B.23}$$

$$\frac{\partial}{\partial y}\int_b^s \tau_{xy}dz \qquad + \frac{\partial b}{\partial y}\tau_{xy}^b \qquad - \frac{\partial s}{\partial y}\tau_{xy}^s \tag{B.24}$$

$$\underbrace{-\tau_{xz}^b}_{-\mu(P-p_w)\frac{u_x}{\|u\|} + \frac{\partial b}{\partial x}\tau_{zz}^b} \qquad + \underbrace{\tau_{xz}^s}_{= -\frac{\partial s}{\partial x}\tau_{zz}^s} - \frac{\partial}{\partial x}\int_b^s \underbrace{P}_{= -\tau_{xx} - \tau_{yy}} dz - \frac{\partial b}{\partial x}P^b + \frac{\partial s}{\partial x}\underbrace{P^s}_{=0}. \tag{B.25}$$

Assuming that τ_{xx}, τ_{yy}, τ_{xy} and τ_{zz} at leading order, this last equations simplifies to

$$\frac{\partial(2H\tau_{xx}+\tau_{yy})}{\partial x}+\frac{\partial\tau_{xy}}{\partial y}-\rho g H\frac{\partial s}{\partial x}-\mu(\rho g H-p_w)\frac{u_x}{\|u\|}=0. \tag{B.26}$$

Finally, if $B^*=H^{-1}\int_b^s B dz$ is the vertical averaged hardness and

$$\nu=\frac{B^*}{2}\left(\frac{1}{2}\left(\frac{\partial u}{\partial x}\right)^2+\frac{1}{2}\left(\frac{\partial v}{\partial y}\right)^2+\frac{1}{2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)^2+\frac{1}{4}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)^2\right)^{\frac{1-n}{2n}}, \tag{B.27}$$

using the viscous form of Glen's law we arrive to the SSA equations

$$2\frac{\partial}{\partial x}\left[\nu H\left(2\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right]+\frac{\partial}{\partial y}\left[\nu H\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]-\rho g H\frac{\partial s}{\partial x}-\tau_c u/\|u\|=0 \tag{SSA (x)}$$

$$2\frac{\partial}{\partial y}\left[\nu H\left(2\frac{\partial v}{\partial y}+\frac{\partial u}{\partial x}\right)\right]+\frac{\partial}{\partial x}\left[\nu H\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]-\rho g H\frac{\partial s}{\partial y}-\tau_c v/\|u\|=0 \tag{SSA (y)}$$

We can also introduce the vertical integrated stress tensor

$$T_{ij}=2\nu H(\dot\epsilon_{ij}+(\dot\epsilon_{xx}+\dot\epsilon_{yy})\delta_{ij}) \tag{B.28}$$

for $i,j=1,2$ and write the SSA equations in a more compact way:

$$\frac{\partial T_{ij}}{\partial x_j}+\tau_i^b=\rho g H\frac{\partial s}{\partial x_i}. \tag{B.29}$$