

Differentiable Programming for Differential Equations: A Review

Facundo Sapienza

Department of Statistics, University of California, Berkeley, USA

`fsapienza@berkeley.edu`

Jordi Bolibar

Univ. Grenoble Alpes, CNRS, IRD, G-INP, Institut des Géosciences de l'Environnement, Grenoble, France

TU Delft, Department of Geosciences and Civil Engineering, Delft, Netherlands

Frank Schäfer

CSAIL, Massachusetts Institute of Technology, Cambridge, USA

Brian Groenke

TU Berlin, Department of Electrical and Computer Engineering, Berlin, Germany

Helmholtz Centre for Environmental Research, Leipzig, Germany

Avik Pal

CSAIL, Massachusetts Institute of Technology, Cambridge, USA

Victor Boussange

Swiss Federal Research Institute WSL, Birmensdorf, Switzerland

Patrick Heimbach

Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, USA

Jackson School of Geosciences, University of Texas at Austin, USA

Giles Hooker

Department of Statistics and Data Science, University of Pennsylvania, USA

Fernando Pérez

Department of Statistics, University of California, Berkeley, USA

Per-Olof Persson

Department of Mathematics, University of California, Berkeley, USA

Christopher Rackauckas

Massachusetts Institute of Technology, Cambridge, USA

JuliaHub, Cambridge, USA

Abstract

The differentiable programming paradigm is a cornerstone of modern scientific computing. It refers to numerical methods for computing the gradient of a numerical model's output. Many scientific models are based on differential equations, where differentiable programming plays a crucial role in calculating model sensitivities, inverting model parameters, and training hybrid models that combine differential equations with data-driven approaches. Furthermore, recognizing the strong synergies between inverse methods and machine learning offers the opportunity to establish a coherent framework applicable to both fields. Differentiating functions based on the numerical solution of differential equations is non-trivial. Numerous methods based on a wide variety of paradigms have been proposed in the literature, each with pros and cons specific to the type of problem investigated. Here, we provide a comprehensive review of existing techniques to compute derivatives of numerical solutions of differential equations. We first discuss the importance of gradients of solutions of differential equations in a variety of scientific domains. Second, we lay out the mathematical foundations of the various approaches and compare them with each other. Third, we cover the computational considerations and explore the solutions available in modern scientific software. Last but not least, we provide best-practices and recommendations for practitioners. We hope that this work accelerates the fusion of scientific models and data, and fosters a modern approach to scientific modelling.

Key words. differentiable programming, sensitivity analysis, differential equations, inverse modelling, scientific machine learning, automatic differentiation, adjoint methods.

Contents

1	Introduction	4
2	Scientific motivation: A domain science perspective	6
2.1	Machine learning	6
2.2	Computational physics and optimal design	7
2.2.1	Computational fluid dynamics	7
2.2.2	Quantum physics	8
2.2.3	Other applications	8
2.3	Geosciences	8
2.3.1	Numerical weather prediction	9
2.3.2	Oceanography	9
2.3.3	Climate science	10
2.3.4	Glaciology	10
2.4	Biology and ecology	10
3	Methods: A mathematical perspective	11
3.1	Preliminaries	13
3.1.1	Numerical solvers for ODEs	13
3.1.2	What to differentiate?	15
3.1.3	Gradient-based optimization	16
3.1.4	Sensitivity matrix	17
3.2	Finite differences	18
3.3	Automatic differentiation	18
3.3.1	Forward mode	19
3.3.1.1	Dual numbers	19
3.3.1.2	Computational graph	20
3.3.2	Reverse mode	20
3.3.3	AD connection with JVPs and VJPs	21
3.4	Complex step differentiation	23
3.5	Symbolic differentiation	23
3.6	Forward sensitivity equations	24
3.7	Discrete adjoint method	25
3.7.1	Adjoint state equations	25
3.7.2	Simple linear system	26
3.8	Continuous adjoint method	28
3.9	Mathematical comparison of the methods	29
3.9.1	Forward AD and complex step differentiation	29
3.9.2	Discrete adjoints and reverse AD	29
3.9.3	Consistency: forward AD and forward sensitivity equations	31
3.9.4	Consistency: discrete and continuous adjoints	32
4	Implementation: A computer science perspective	33
4.1	Direct methods	34
4.1.1	Finite differences	34
4.1.2	Automatic differentiation	34

4.1.2.1	Forward AD based on dual numbers	34
4.1.2.2	Reverse AD based on computational graph	36
4.1.2.3	Discrete checkpointing	37
4.1.2.4	When AD is algorithmically correct but numerically wrong	37
4.1.2.5	Differentiation with sparsity	38
4.1.3	Complex step differentiation	39
4.2	Solver-based methods	40
4.2.1	Forward sensitivity equation	40
4.2.1.1	Computing JVPs and VJPs inside the solver	41
4.2.2	Adjoint methods	42
4.2.2.1	Discrete adjoint method	42
4.2.2.2	Continuous adjoint method	43
4.2.2.3	Continuous checkpointing	45
4.2.2.4	Solving the quadrature	45
5	Generalizations	46
5.1	Higher-order ODEs	46
5.2	Partial differential equations	46
5.3	Chaotic systems	48
6	Recommendations	49
7	Conclusions	52
	Appendices	55
A	Supplementary code	55
	References	56

*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.*

1 Introduction

Models based on differential equations (DEs), including ordinary differential equations (ODEs) and partial differential equations (PDEs), play a central role in describing the behaviour of dynamical systems in applied, natural, and social sciences. For instance, DEs are used for the modelling of the dynamics of the atmospheric and ocean circulation in climate science, for the modelling of ice or mantle flow in solid Earth geophysics, for the modelling of the spatio-temporal dynamics of species abundance in ecology, and for the modelling of cancer cell evolution in biology. For centuries, scientists have relied on theoretical and analytical methods to solve DEs. Allowing to approximate the solutions of large, nonlinear DE-based models, numerical methods and computers have lead to fundamental advances in the understanding and prediction of physical, biological, and social systems, among others (Dahlquist 1985; Hey et al. 2009; R  de et al. 2018).

Quantifying how much the output of a DE-based model changes with respect to its input parameters is fundamental to many scientific computing and machine learning applications, including optimization, sensitivity analysis, Bayesian inference, inverse methods, and uncertainty quantification, among many (Razavi et al. 2021). Mathematically, quantifying this change involves evaluating the gradient of the model, i.e., calculating a vector whose components are the partial derivatives of the model evaluated at the model parameter values. In sensitivity analysis, gradients are crucial for comprehending the relationships between model inputs and outputs, assessing the influence of each parameter, and evaluating the robustness of model predictions. In optimization and inverse modelling, where the goal is to fit models to data and/or invert for unknown or uncertain parameters, gradient-based methods are more efficient at finding a minimum and converge faster to them than gradient-free methods. In Bayesian inference, gradient-based sampling strategies are better at estimating the posterior distribution than gradient-free methods (Neal et al. 2011). Therefore, accurately determining model gradients is essential for robust model understanding and effective data assimilation that leverage strong physical priors while offering flexibility to adapt to observations. This is very appealing in fields such as computational physics, geophysics, and biology, to mention a few, where there is a broad literature on DE-based models. The techniques used to compute these gradients fall within the framework of differentiable programming.

Differentiable programming (DP) refers to a set of techniques and software tools for computing gradients/sensitivities of a model’s output, evaluated using a computer program, with respect to the model’s input variables or parameters (Blondel et al. 2024; Innes et al. 2019; Shen et al. 2023). Arguably, the most celebrated DP method is automatic differentiation. The set of tools known as automatic or algorithmic differentiation (AD) aims to compute derivatives of a model rendered on a computer by 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. These operations are individually easy to differentiate, and their composition can be easily differentiated using the chain rule (Giering et al. 1998). During the last decades, reverse mode AD, also known as backpropagation, has enabled the fast growth of deep learning by efficiently computing gradients of neural networks (Griewank 2012). Some authors have recently suggested DP 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). More broadly than AD, DP tools for DE-models further include forward sensitivity and adjoint methods that compute the gradient by relying on an auxiliary set of differential equations.

The development of DP for DE-based models has a long tradition across different scientific communities. In statistics, gradients of the likelihood function of DE-based models enable inference on the model parameters (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, 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 (Giles et al. 2000). In recent years, there has been an increasing interest in designing machine learning workflows that include constraints in the form of DEs and are trained using gradient descent techniques. This emerging sub-field is usually referred as physics-based or physics-informed machine learning (Karniadakis et al. 2021; Thuerey et al. 2021; Vadyala et al. 2022).

The need for model gradients is even more critical as the total number of parameters and the expressivity of the model increases, especially when dealing with highly non-linear processes. In such circumstances, the curse of dimensionality renders gradient-free optimization and sampling methods computationally intractable. This is the case in inverse methods (Ghattas et al. 2021; Tarantola 2007) and in machine learning applications (LeCun et al. 2015), where highly parametrized regressor functions (e.g., neural networks) are used to approximate unknown non-linear function. Furthermore, for stochastic forward models, the intractability of the likelihood function represents a major challenge for statistical inference. The integration of DP has provided new tools for resolving complex simulation-based inference problems (Cranmer et al. 2020).

Computing gradients of functions, represented by DE-based simulation codes, with respect to their (high-dimensional) inputs is challenging due to the complexities in both the mathematical framework and the software implementation involved. Except for a small set of particular cases, most DEs require numerical methods to approximate their solution, which means that solutions cannot be differentiated analytically. Furthermore, numerical solutions introduce approximation errors. These errors can be propagated to the computation of the gradient, leading to inaccurate or inconsistent gradient values. Additional to these complexities, the broad family of numerical methods, each one of them with different advantages depending on the DE (Hairer et al. 2008; Wanner et al. 1996), means that the tools developed to compute gradients need to be universal enough in order to be applied to all or at least to a large set of them.

There exists a large family of methods to compute derivatives of DE-based models. The differences between methods to compute derivatives arise both from their mathematical formulation, numerical stability, and their computational implementation. They can be roughly classified as continuous (differentiate-then-discretize) or discrete (discretize-then-differentiate) and forward or reverse. Different methods guarantee different levels of accuracy, have different computational complexity, and require different trade-offs between run time and memory usage. These properties further depend of the total number of parameters and size of the DE. Despite their independent success, integrating DP with DE-based models remains a significant challenge in high-performance scientific computing (Naumann 2011).

This paper presents a comprehensive review of methods for calculating derivatives of the numerical solution of differential equations, with a focus on efficiently computing gradients. We review differentiable programming methods for differential equations from three different perspectives: a domain science perspective (Section 2), a mathematical perspective (Section 3) and a computer science perspective (Section 4). In Section 2 we introduce some of the applications of DP for the modelling of complex systems in the natural and social sciences. In Section 3 we present a coherent mathematical framework to understand the theoretical differences between the DP methods. In Section 4 we show how these methods can be computationally implemented and what are their numerical advantages and disadvantages. For simplicity, all the methods introduced in Sections 3 and 4 focus exclusively on first-order ODEs. How these methods generalize to other DE-based models, including PDEs, is discussed in Section 5. We conclude the paper with a series of recommendations in Section 6. By providing a common framework across all methods and applications, we hope to

facilitate the development of scalable, practical, and efficient differentiable DE-based models.

2 Scientific motivation: A domain science perspective

Mechanistic (or process-based) models play a central role in a wide range of scientific disciplines. They consist of mathematical descriptions of physical mechanisms that include 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 DEs. Together with the numerical methods to approximate their solutions, DEs have led to fundamental advances in the understanding and prediction of physical and biological systems.

DEs usually depend on inputs or parameters that change the obtained solutions. While direct or forward modelling usually refers to understanding how these parameters map into solutions or observations of the DE-based model, the overarching goal of inverse modelling is to find a set of optimal model parameters that minimizes an objective or cost function quantifying the misfit between observations and the simulated state. The constrained optimization problem is transformed into an unconstrained problem using the Lagrangian formulation, also referred to as the adjoint method (Givoli 2021; Vadlamani et al. 2020). The corresponding adjoint model computes the gradient of the objective function with respect to all inputs. Gradient-based nonlinear optimization enables to invert for optimal values of the unknown or uncertain inputs. Depending on the nature of the inversion, we may distinguish between the following cases:

- **Initial conditions.** Inverting for uncertain initial conditions, which, when integrated using the model, leads to an optimal match between the observations and the simulated state (or diagnostics); variants thereof are used for optimal forecasting.
- **Boundary conditions.** Inverting for uncertain surface (e.g., interface fluxes), bottom (e.g., bed properties), or lateral (e.g., open boundaries of a limited domain) boundaries, which, when used in the model, produce an optimal match of the observations. Variants thereof are used in tracer or boundary (air-sea) flux inversion problems, e.g., related to the global carbon cycle.
- **Model parameters.** Inverting for uncertain model parameters amounts to an optimal model calibration problem. As a *learning of optimal parameters from data* problem, it is the closest to machine learning applications. Parametrization is a special case of parameter inversion, where a parametric function (e.g., a neural network) is used to approximate processes.

Besides the use of sensitivity methods for optimization, inversion, estimation, or learning, gradients have also proven powerful tools for computing comprehensive sensitivities of quantities of interest; computing optimal perturbations (in initial or boundary conditions) that lead to maximum, non-normal amplification of specific norms of interest; and characterizing and quantifying uncertainties by way of second derivative (Hessian) information. The availability of second derivatives (Hessian) further helps to improve the convergence rates of optimization algorithms.

In the following, we present selected examples belonging to a wide range of scientific communities where DP techniques have been used for the modelling of systems described using DEs.

2.1 Machine learning

In recent years the use of machine learning methods has become more popular in many scientific domains (Borowiec et al. 2022; Lai et al. 2024; Meuwly 2021; Pichler et al. 2023; Rasp et al.

2018). By learning nonlinear patterns from large datasets at multiple levels of abstraction (LeCun et al. 2015), these methods are highly flexible with respect to inputs and outputs required, and can be exploited by many different domain-specific problems. However, the use of machine learning models for prediction has been heavily criticized, as they critically assume that patterns contained in observed data will repeat in the future, which may not be the case (Barnosky et al. 2012; Dormann 2007). In contrast to purely statistical models, the process knowledge embedded in the structure of mechanistic models renders them more robust for predicting dynamics under different conditions (Barnosky et al. 2012). The fields of mechanistic modelling and statistical modelling have mostly evolved independently due to several reasons (Zdeborová 2020). On the one hand, domain scientists have often been reluctant to learning about machine learning methods, judging them as opaque black boxes, unreliable, and not respecting domain-established knowledge (Coveney et al. 2016). On the other hand, the field of machine learning has mainly been developed around data-driven applications, without including any a priori physical knowledge. 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. This sub-field, usually known as physics-informed machine learning, refers to 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 DEs (Karniadakis et al. 2021). It includes methods that numerically solve DEs, such as physics-informed neural networks (Raissi et al. 2019), biology-informed neural networks (Lagergren et al. 2020; Yazdani et al. 2020), NeuralPDEs (Zubov et al. 2021) and mesh-free methods for solving high-dimensional PDEs (Boussange et al. 2023). On the other hand, there has been an increased interest in augmenting DE-based models by embedding a rich family of parametric functions (e.g., neural networks) inside the DE. This approach is known as such as universal differential equations (Dandekar et al. 2020; Rackauckas et al. 2020), which also include the case of neural ordinary differential equations (Chen et al. 2018) and neural stochastic differential equations (Li et al. 2020b).

2.2 Computational physics and optimal design

There is a long tradition of computational physics models based on adjoint methods and AD pipelines, where sensitivity methods have been used for optimal design and optimal control since the 1960s (Lions 1971). These models are often based on PDEs and are applied in various fields to improve engineering designs or model parameters with respect to some objective function. The models can involve thousands of parameters, and they require efficient derivative calculations for the use of gradient-based optimizers such as quasi-Newton methods (Nocedal et al. 1999). Both discrete and continuous adjoints methods, which we will introduce in Sections 3.7 and 3.8, respectively, have been used extensively, each having different benefits depending on the application.

2.2.1 Computational fluid dynamics

DP methods, including AD and adjoint methods, have been crucial in advancing computational fluid dynamics (CFD) applications (Kenway et al. 2019). These techniques have been employed in optimizing the aerodynamics of aircraft for drag reduction, or for weight reduction in aircraft design, leading to significant fuel savings and enhanced performance (Jameson 2003). In aeroacoustic designs, adjoint methods can be used to minimize noise emissions (Freund 2010). The objective function in these applications typically relates to performance metrics or cost considerations, and a wide array of design parameters can be optimized. Entire geometries can be parameterized for *shape optimization*, enabling the refinement of complex structures like airfoils, which are critical

for aerodynamic efficiency. Pironneau introduced fundamental methods for shape optimization in fluid mechanics (Pironneau 1974), and Jameson developed adjoint-based optimization methods that significantly improved aerodynamic designs (Jameson 1988). For comprehensive reviews of shape optimization using adjoint methods, we refer to (Giles et al. 2000) and (Mohammadi et al. 2009). Adjoint methods have also been used for topology optimization (Allaire et al. 2014).

For aerospace applications, adjoint methods have been used to design supersonic aircraft, enhancing performance and reducing sonic boom impacts (Fike 2013; Hu 2010). Entire aircraft configurations have also been optimized using adjoint methods (Chen et al. 2016). Beyond aerospace, other significant applications include optimizing ship hull designs to reduce drag and improve fuel efficiency (Kröger et al. 2018), the aerodynamic shaping of cars to enhance speed and stability (Othmer 2014), and the design of wind turbines to maximize energy capture and structural resilience (Dhert et al. 2017).

2.2.2 Quantum physics

Quantum optimal control has applications spanning a broad spectrum of quantum systems. Optimal control methods have been used to optimize pulse sequences, enabling the design of high-fidelity quantum gates and the preparation of complex entangled quantum states (Koch et al. 2022). Typically, the objective is to maximize the fidelity to a target state or unitary operation, accompanied by additional constraints or costs specific to experimental demands. The predominant control algorithms are gradient-based optimization methods, such as gradient ascent pulse engineering (GRAPE), and rely on the computation of derivatives for solutions of the ODEs modeling the time evolution of the quantum system. In cases where the analytical calculation of a gradient is impractical, numerical evaluation using AD becomes a viable alternative (Abdelhafez et al. 2020, 2019; Goerz et al. 2022; Jirari 2009, 2019; Leung et al. 2017; Schäfer et al. 2020). Specifically, AD streamlines the adjustment to diverse objectives or constraints, and its efficiency can be enhanced by employing custom derivative rules for the time propagation of quantum states as governed by solutions to the Schrödinger equation (Goerz et al. 2022). Moreover, sensitivity methods facilitate the design of feedback control schemes necessitating the differentiation of solutions to stochastic differential equations (Schäfer et al. 2021a).

2.2.3 Other applications

Adjoint methods have also been applied successfully to a wide range of other computational physics problems. In particle physics, they enable precise parameter estimation and simulation improvements (Dorigo et al. 2022). In quantum chemistry, adjoint methods can be used to optimize molecular structures and reaction pathways (Arrazola et al. 2021). The design of nanophotonic devices, such as photonic crystals and waveguides, has been significantly advanced through these techniques (Molesky et al. 2018). Electromagnetic applications, including the optimization of antenna designs and microwave circuits, benefit from the fine-tuning capabilities provided by adjoint methods (Georgieva et al. 2002). Stellarator coil design for nuclear fusion reactors is another important area, where adjoint methods contribute to optimizing magnetic confinement configurations (McGreivy et al. 2021). Sensitivity analysis methods are also very popular in topological and structural design (Min et al. 1999; Van Keulen et al. 2005).

2.3 Geosciences

Many geoscientific phenomena are governed by conservation laws along with a set of empirical constitutive laws and subgrid-scale parametrization schemes. Together, they enable efficient description

of the system’s spatio-temporal evolution in terms of a set of PDEs. Example are geophysical fluid dynamics (Vallis 2016), describing geophysical properties of many Earth system components, such as the atmosphere, ocean, land surface, and glaciers. In such models, calibrating model parameters is extremely challenging, due to observational data being sparse in both space and time, heterogeneous, and noisy; and computational models involving high-dimensional parameter spaces, often on the order of $O(10^3) - O(10^8)$. Moreover, many existing mechanistic models can only partially describe observations, with many detailed physical processes being ignored or poorly parameterized.

2.3.1 Numerical weather prediction

Numerical weather prediction (NWP) is among the most prominent fields where adjoint methods have played an important role (Errico 1997). Adjoint methods were introduced to infer initial conditions that minimize the misfit between simulations and weather observations (Courtier et al. 1987; Lewis et al. 1985; Talagrand et al. 1987), with the value of second-derivative information also being recognized (Dimet et al. 2002). This led to the development of the four-dimensional variational (4D-Var) technique at the European Centre for Medium-Range Weather Forecasts (ECMWF) as one of the most advanced data assimilation approaches (Rabier et al. 2000; Rabier et al. 1992), and which contributed substantially to the *quiet revolution* in NWP (Bauer et al. 2015). Related, within the framework of transient non-normal amplification or optimal excitation (Farrell 1988; Farrell et al. 1996), the adjoint method has been used to infer patterns in initial conditions that over a finite time contribute to maximum uncertainty growth in forecasts (Buizza et al. 1995; Palmer et al. 1994) and to infer the so-called Forecast Sensitivity-based Observation Impact (FSOI) (Langland et al. 2004). Except for early research applications (Park et al. 1996; Park et al. 2000) and for experimental purposes (Giering et al. 2006), AD has not been widely used in the development of adjoint models in NWP. Instead, the adjoint code has been, for the most part, derived and implemented manually.

2.3.2 Oceanography

The recognition of the benefit of adjoint methods for use in data assimilation in the ocean coincided roughly with that in weather prediction (Thacker 1988, 1989; Thacker et al. 1988; Tziperman et al. 1992a,b; Tziperman et al. 1989). An important detail is that their work already differed from the 4D-Var problem of NWP (Section 2.3.1) in that sensitivities were computed not only with respect to initial conditions but surface boundary conditions. Similar to the work on calculating singular vectors in the atmosphere, the question of El Niño (Moore et al. 1997a,b) and Atlantic Meridional Overturning (AMOC) (Zanna et al. 2012; Zanna et al. 2011, 2010) predictability invited model-based singular vector computations in ocean models. More recent data assimilation frameworks with fully hand-written adjoint codes include the NEMO model (Vidard et al. 2015; Weaver et al. 2003) and the ROMS model (Moore et al. 2011, 2004).

The consortium for Estimating the Circulation and Climate of the Ocean (ECCO) (Stammer et al. 2002) set out in around 1999 to develop a parameter and state estimation framework where the adjoint model is generated using source-to-source AD (Heimbach et al. 2005; Marotzke et al. 1999). Rigorous exploitation of AD enabled extensions to vastly improved model numerics (Forget et al. 2015) and coupling to biogeochemistry (Dutkiewicz et al. 2006), sea-ice (Heimbach et al. 2010), and sub-ice shelf cavities (Goldberg et al. 2020; Heimbach et al. 2012). Unlike NWP-type 4D-Var, it also enabled extension to the problem of parameter calibration from observations (Ferreira et al. 2005; Liu et al. 2012; Stammer 2005). Arguably, this work heralded much of today’s efforts in *online learning* of parameterization schemes. The desire to make AD for Earth system models written in Fortran (to date the vast majority) has spurred the development of AD tools with powerful reverse

modes, both commercial (Giering et al. 1998, 2006) and open-source (Gaikwad et al. 2023, 2024; Hascoet et al. 2013; Utke et al. 2008).

2.3.3 Climate science

The same goals that have driven the use of sensitivity information in NWP (optimal initial conditions for forecasts) or ocean science (state and parameter estimation) apply in the world of climate modeling. The recognition that good initial conditions (e.g., such that are closest to the real or observed system) will lead to improved forecasts on subseasonal, seasonal, interannual, or even decadal time scales has driven major community efforts (Meehl et al. 2021). However, there has been a lack so far in exploiting the use of gradient information to achieve optimal initialization for coupled Earth system models (Frolov et al. 2023). One conceptual challenge is the presence of multiple timescales in the coupled system and the utility of gradient information beyond many synoptic time scales in the atmosphere and ocean (Lea et al. 2000, 2002). Nevertheless, efforts are underway to enable adjoint-based parameter estimation of coupled atmosphere-ocean climate models, with AD again playing a crucial role in generating the corresponding adjoint model (Blessing et al. 2014; Lyu et al. 2018; Stammer et al. 2018). Complementary, recognizing the power of DP, efforts are also targeting the development of neural atmospheric general circulation models in JAX, which combine a differentiable dynamical core with neural operators as surrogate models of unresolved physics (Kochkov et al. 2023).

2.3.4 Glaciology

Due to the difficulty of taking direct measurements of internal and basal rheological processes of glaciers and ice sheets, inverse methods based on adjoint models have been widely used to study them, following the pioneering work by (MacAyeal 1992) in the 1990s. Since then, the adjoint method has been applied to many different studies investigating parameter and state estimation (Goldberg et al. 2013; Vieli et al. 2006), ice volume sensitivity to basal, surface and initial conditions (Heimbach et al. 2009), inversion of initial conditions (Mosbeux et al. 2016) or inversion of basal friction (Morlighem et al. 2013; Petra et al. 2012). These studies either derived the adjoint with a manual implementation or combined AD with hand-written adjoint solvers. The use of AD has become increasingly widespread in glaciology, paving the way for more complex modelling frameworks (Gaikwad et al. 2023; Hascoët et al. 2018). The use of second-derivative (Hessian) information has also been recognized as a powerful approach for conducting rigorous uncertainty quantification in the context of ice sheet parameter inversion (Isaac et al. 2015; Petra et al. 2014). For a recent comprehensive review of data assimilation in ice sheet modeling, with emphasis on adjoint methods, see (Morlighem et al. 2023).

Recently, DP has also facilitated the development of hybrid frameworks, combining numerical methods with data-driven models by means of universal differential equations (Bolibar et al. 2023). Alternatively, some other approaches have dropped the use of numerical solvers in favour of physics-informed neural networks, exploring the inversion of rheological properties of glaciers (Wang et al. 2022) and to accelerate ice thickness inversions and simulations by leveraging GPUs (Jouvet 2023; Jouvet et al. 2021).

2.4 Biology and ecology

DE-based models have been broadly used in biology and ecology to model neural firing (Hodgkin et al. 1952), the dynamics of genes and alleles (Page et al. 2002), immune and disease processes (Colijn et al. 2006), subcellular functions (Brown et al. 2003), the ecological and evolutionary dynamics

of biological units from bacteria to ecological communities (Åkesson et al. 2021; Boussange et al. 2023, 2022; Chalmandrier et al. 2021; Gábor et al. 2015; Lion 2018; van den Berg et al. 2022; Villa et al. 2021), and biomass and energy fluxes and transformation at ecosystem levels (Franklin et al. 2020; Geary et al. 2020; Schartau et al. 2017; Weng et al. 2015). Parameters in DE-based models are often estimated from direct laboratory experiments (e.g. (Hodgkin et al. 1952)) although this process is costly and difficult (Schartau et al. 2017), and may result in simulations failing in capturing real biological dynamics (Watts 2001). Alternatively, inverse modelling methods also have a substantial history for both parameter estimation (Ding 2000; Fussmann et al. 2000; Ramsay et al. 2017; Ramsay et al. 2007; Schartau et al. 2017) and model selection (Alsos et al. 2023; Johnson et al. 2004; Pantel et al. 2023; Zhang et al. 2015). When parameters are inferred along with their uncertainties, they can be interpreted to better understand the strengths and effects of the processes under consideration (Curtsdotter et al. 2019; Godwin et al. 2020; Higgins et al. 2010; Pontarp et al. 2019). However, in high-dimensional models parameters are often non-identifiable (Transtrum et al. 2011). Model selection, which does not require parameter identifiability, involves deriving candidate models that embed competing hypotheses about causal processes and computing the relative evidence for each model given the data to discriminate between hypotheses (Alsos et al. 2023; Johnson et al. 2004). The computation of the most likely model parameter values, or the evaluation of different model supports, critically involves sensitivity methods that must effectively handle the typically large number of parameters and the nonlinearities of biological models (Gábor et al. 2015; Transtrum et al. 2011). Sensitivity methods also play a crucial role in assessing model fit quality, specifying system states, detecting stochastic noise (Hooker 2009; Hooker et al. 2015; Liu et al. 2023), and designing experiments to optimize parameter estimation precision (Bauer et al. 2000).

While inverse modeling based on AD in biology has traditionally overlooked, its potential has recently been highlighted (Alsos et al. 2023; Frank 2022). New approaches involving AD are increasingly being proposed to accommodate the specific requirements of biological and ecological models (Boussange et al. 2024; Lagergren et al. 2020; Paredes et al. 2023; Yazdani et al. 2020). Given that key processes are often not accurately represented in biological models (Chalmandrier et al. 2021; Hartig et al. 2012; Schartau et al. 2017), hybrid approaches that integrate data-driven parameterization of specific components of DE-based models are particularly relevant (Boussange et al. 2024; Cao et al. 2008; Chen et al. 2017; Dai et al. 2022; Paul et al. 2011; Ramsay 1996; Rasp et al. 2018). Sensitivities also play an important role in assessment of fit quality including specification of system states and detecting the presence of stochastic noise (Hooker 2009; Hooker et al. 2015; Liu et al. 2023) and in the design of experiments to optimize the precision of parameter estimates (Bauer et al. 2000). The increasing availability of biological datasets, driven by advancements in monitoring technologies such as paleo-time series (Alsos et al. 2023), environmental DNA (Ruppert et al. 2019), remote sensing (Jetz et al. 2019), bioacoustics (Aide et al. 2013), and citizen observations (GBIF: The Global Biodiversity Information Facility 2022), presents additional opportunities to enhance mechanistic models with data-driven components.

3 Methods: A mathematical perspective

There is a large family of methods for computing gradients of functions based on solutions of DEs. Depending on the number of parameters and the characteristics of the DE (e.g., level of stiffness), they have different mathematical, numerical, and computational advantages. These methods can be roughly classified as follows:

- *Continuous vs discrete methods*

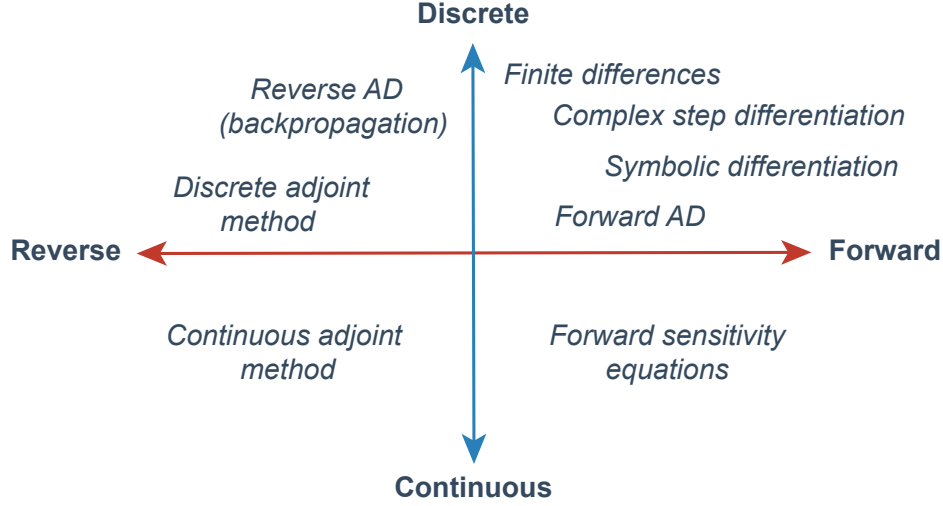


Figure 1: Schematic representation of the different methods available for differentiation involving differential equation solutions. These can be classified depending if they find the gradient by solving a new system of differential equations (continuous) or if instead they manipulate unit algebraic operations (discrete). Additionally, these methods can be categorized depending if sensitivities are propagated from input to output (forward) or from output to input (reverse).

► *Forward vs reverse methods*

Figure 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 function of the solution of a differential equation, one needs to derive both a mathematical expression for the gradient (the differentiation step) and solve the differential 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 refer to 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 the numerical solver (AD, symbolic differentiation, discrete adjoint method). It is worth noting that although both approaches are classified as discrete methods, their numerical properties are quite different. In the case of *continuous* methods, a new set of DEs is derived that allow the calculation of the desired gradient, namely the sensitivity (forward sensitivity equations) or the adjoint (continuous adjoint method) of the system. When comparing discrete to continuous methods, we are focusing, beyond computational efficiency, 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 (Eberhard et al. 1996; Walther 2007). On the other side, continuous methods lead to consistency error calculation of the sensitivity method (Keulen et al. 2005).

The distinction between *forward* and *reverse* regards whether sensitivities are computed for intermediate variables with respect to the input variable or parameter to differentiate (forward) or, on the contrary, we compute the sensitivity of the output variable with respect to each intermediate variable by defining a new adjoint variable (reverse). Mathematically speaking, this distinction translated to the fact that forward methods compute directional derivatives by mapping sequential

mappings between tangent spaces, while reverse methods apply sequential mappings between co-tangent spaces from the direction of the output variable to the input variable (Section 3.3.3). In all forward methods the DE is solved sequentially and simultaneously with the directional derivative during the forward pass of the numerical solver. On the contrary, reverse methods compute the gradient by solving a new problem that moves in the opposite direction as the original numerical solver. In DE-based models, intermediate variables correspond to intermediate solutions of the DE. In the case of ODEs and time-dependant PDEs, most numerical methods solve the DE by progressively moving forward in time, meaning that reverse methods solve for the gradient moving backwards in time. In other words, forward methods compute directional derivatives as they simultaneously solve the original DE, while reverse methods compute adjoints as they solve the problem from output to input.

As discussed in the following sections, forward methods are very efficient for problems with a small number of parameters we want to differentiate with respect to. Conversely, reverse methods, though more efficient for a large number of parameters, incur greater memory costs and computational overhead which need 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, forward sensitivity equations, symbolic differentiation) compute the full sensitivity of the differential equation, which can be computationally expensive or intractable 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).

The rest of this section is organized as follows. We first introduce some basic mathematical notions to facilitate the discussion of the DP methods (Section 3.1). We then mathematically formalize each of the methods listed in Figure 1. We finally discuss the mathematical foundations of these methods in 3.9 with a comparison of some mathematical foundations of these methods.

3.1 Preliminaries

Consider the first-order ODE given by

$$\frac{du}{dt} = f(u, \theta, t) \quad (1)$$

subject to the initial condition $u(t_0) = u_0$, where $u \in \mathbb{R}^n$ is the unknown solution vector of the ODE, $f : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R} \mapsto \mathbb{R}^n$ is a function that depends on the state u , $\theta \in \mathbb{R}^p$ is a vector parameter, and $t \in [t_0, t_1]$ refers to time. Here, n denotes the size of the ODE and p the number of parameters. Except for a minority of functions $f(u, \theta, t)$, solutions to Equation (1) need to be computed using numerical solvers.

3.1.1 Numerical solvers for ODEs

Numerical solvers for the solution of ODEs or initial value problems 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 , s -stage Runge-Kutta methods are defined by generalizing

numerical integration quadrature rules as follows

$$\begin{aligned}
u^{m+1} &= u^m + \Delta t_m \sum_{i=1}^s b_i k_i \\
k_i &= f \left(u^m + \sum_{j=1}^s a_{ij} k_j, \theta, t_m + c_i \Delta t_m \right) \quad i = 1, 2, \dots, s.
\end{aligned} \tag{2}$$

where $u^m \approx u(t_m)$ approximates the solution at time t_m , $\Delta t_m = t_{m+1} - t_m$, and a_{ij} , b_i , and c_j are scalar coefficients with $i, j = 1, 2, \dots, s$, 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 fully implicit otherwise. Different choices of the number of stages s and coefficients give different orders of convergence of the numerical scheme (Butcher 2001; Butcher et al. 1996).

In contrast, multi-step linear solvers are of the form

$$\sum_{i=0}^{k_1} \alpha_{mi} u^{m-i} = \Delta t_m \sum_{j=0}^{k_2} \beta_{mj} f(u^{m-j}, \theta, t_{m-j}) \tag{3}$$

where α_{mi} and β_{mj} are numerical coefficients (Hairer et al. 2008). In most cases, including the Adams methods and backwards differentiation formulas (BDF), we have the coefficients $\alpha_{mi} = \alpha_i$ and $\beta_{mj} = \beta_j$, meaning that the coefficient do not depend on the iteration m . Notice that multi-step 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 $\beta_{m0} = 0$ and are easy to solve by direct iterative updates. For implicit methods, the usually non-linear equation

$$g_m(u^m; \theta) = u^m - \Delta t_m \beta_{m0} f(u^m, \theta, t_m) - \bar{\alpha}_m = 0, \tag{4}$$

with $\bar{\alpha}_m$ a computed coefficient that includes the information of all past iterations, can be solved using predictor-corrector methods (Hairer et al. 2008) or iteratively using Newton's method and preconditioned Krylov solvers at each nonlinear iteration (Hindmarsh et al. 2005).

When choosing a numerical solver for differential equations, one crucial factor to consider is the stiffness of the equation. Stiffness encompasses various definitions, reflecting its historical development and different types of instabilities (Dahlquist 1985). Two definitions are noteworthy

- 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; Wanner et al. 1996), also characterized by the phenomena of increasing oscillations (Dahlquist 1985).

Stability properties can be achieved by different means, for example, by using implicit methods or stabilized explicit methods, such as Runge-Kutta-Chebyshev (Der Houwen et al. 1980; Wanner et al. 1996). When using explicit methods, smaller timesteps may be required to guarantee stability.

Numerical solvers usually estimate internally a scaled error computed as

$$\text{Err}_{\text{scaled}}^m = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{\text{err}_i^m}{\text{abstol} + \text{reltol} \times M_i^m} \right)^2}, \tag{5}$$

with `abstol` and `reltol` the adjustable absolute and relative solver tolerances, respectively, M is the maximum expected value of the numerical solution, and err^m is an estimation of the numerical error at step m (Hairer et al. 2008; Rackauckas et al. 2016). Estimations of the local error err^m can be based on two approximation to the solution based on embedded Runge-Kutta pairs (Hairer et al. 2008; Ranocha et al. 2022), or in theoretical upper bounds provided by the numerical solver. In the first case, common choices for these include $M_i^m = \max\{u_i^m, \hat{u}_i^m\}$ and $\text{err}_i^m = u_i^m - \hat{u}_i^m$ with u^m and \hat{u}^m two different approximations for $u(t_m)$, but these can vary between solvers.

Modern solvers include stepsize controllers that pick Δt_m as large as possible to minimize the total number of steps while preventing large errors by keeping $\text{Err}_{\text{scaled}}^m \leq 1$. One of the most used methods to archive this is the proportional-integral controller (PIC) that updates the stepsize according to

$$\Delta t_m = \eta \Delta t_{m-1} \quad \eta = w_m^{\beta_1/q} w_{m-1}^{\beta_2/q} w_{m-2}^{\beta_3/q} \quad (6)$$

with $w_m = 1/\text{Err}_{\text{scaled}}^m$ the inverse of the scaled error estimates; β_1 , β_2 , and β_3 numerical coefficients defined by the controller; and q the order of the numerical solver (Ranocha et al. 2022; Wanner et al. 1996). If the stepsize Δt_m proposed in Equation (6) to update from u^m to u^{m+1} does not satisfy $\text{Err}_{\text{scaled}}^{m+1} \leq 1$, a new smaller stepsize is proposed. When $\eta < 1$ (which is the case for simple controllers with $\beta_2 = \beta_3 = 0$), Equation (6) can be used for the local update. It is also common to restrict $\eta \in [\eta_{\min}, \eta_{\max}]$ so the stepsize does not change abruptly (Hairer et al. 2008).

3.1.2 What to differentiate?

In most applications, the need for differentiating the solution of ODEs stems from the need to obtain the gradient of a function $L(\theta) = L(u(\cdot, \theta))$ with respect to the parameter θ , where L can denote:

- **A loss function or an 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

$$L(\theta) = \frac{1}{2} \|u(t_1; \theta) - u^{\text{target}}(t_1)\|_2^2, \quad (7)$$

where $u^{\text{target}}(t_1)$ is the desired target observation at some later time t_1 , and $\|\cdot\|_2$ is the Euclidean norm. More generally, we can evaluate the loss function at points of the time series for which we have observations,

$$L(\theta) = \frac{1}{2} \sum_{i=1}^N w_i \|u(t_i; \theta) - u^{\text{target}}(t_i)\|_2^2. \quad (8)$$

with w_i some arbitrary non-negative weights. More generally, misfit functions used in optimal estimation and control problems are composite maps from the parameter space θ via the model's state space (in this case, the solution $u(t; \theta)$) to the observation space defined 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 generalizes to

$$L(\theta) = \frac{1}{2} \sum_{i=1}^N w_i \|H(u(t_i; \theta)) - y^{\text{target}}(t_i)\|_2^2. \quad (9)$$

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

$$L(u(\cdot; \theta)) = \int_{t_0}^{t_1} h(u(t; \theta), \theta) dt, \quad (10)$$

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.

- **The likelihood function or posterior probability.** From a statistical and physical 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 noisy observation of the state $u(t_i; \theta)$. If $p(Y|t, \theta)$ is the probability distribution of $Y = (y_1, y_2, \dots, y_N)$, the maximum likelihood estimator (MLE) of θ is defined as

$$\theta^{\text{MLE}} = \underset{\theta}{\operatorname{argmax}} \ell(Y|\theta) = \prod_{i=1}^N p(y_i|\theta, t_i). \quad (11)$$

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 (9) (Hastie et al. 2009),

$$\theta^{\text{MLE}} = \underset{\theta}{\operatorname{argmin}} \{-\log \ell(Y|\theta)\} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^N \frac{1}{2\sigma_i^2} \|y_i - H(u(t_i; \theta))\|_2^2. \quad (12)$$

A Bayesian formulation of equation (12) would consist in deriving a point estimate θ^{MLE} , the posterior mean of the maximum a posteriori (MAP), based on the posterior distribution for θ following Bayes theorem as $p(\theta|Y) = p(Y|\theta)p(\theta)/p(Y)$, where $p(\theta)$ is the prior distribution (Murphy 2022). In most realistic cases, the posterior distribution is 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 Monte Carlo (Betancourt 2017).

- **A quantity of interest.** In some applications we are interested in quantifying how the solution of the differential equation changes as we vary the parameter values; or more generally when it returns the value of some variable that is a function of the solution of a differential equation. The latter corresponds to 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. 2000; 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.

3.1.3 Gradient-based optimization

In the context of optimization, the goal is to find the parameter θ that is a minimizer of $L(\theta)$. There exists a broad literature of optimization methods based on gradients, including gradient descent and its many variants (Ruder 2016). Gradient-based methods tend to outperform gradient-free optimization schemes when $1 \ll p$, as they are not prone to the curse of dimensionality (Schartau

et al. 2017). In the case of gradient descent, the parameter θ is updated based on the iterative procedure given by

$$\theta^{m+1} = \theta^m - \alpha_m \frac{dL}{d\theta}(\theta^m), \quad (13)$$

with α_m some choice of the stepsize and some initialization $\theta^0 \in \mathbb{R}^p$. A direct implementation of gradient descent following Equation (13) is prone to converge to a local minimum and slows down in a neighborhood of saddle points. To address these issues, variants of this scheme employing more advanced updating strategies have been proposed, including Newton-type methods (Xu et al. n.d.), quasi-Newton methods, acceleration techniques (Muehlebach et al. 2021), and natural gradient descent methods (Nurbekyan et al. 2023). For instance, ADAM is an adaptive, momentum-based algorithm that stores the parameter update at each iteration, and determines the next update as a linear combination of the gradient and the previous update (Kingma et al. 2014). ADAM been widely adopted to train highly parametrized neural networks (up to the order of 10^8 parameters (Vaswani et al. 2017)). Other widely employed algorithms are the Broyden–Fletcher–Goldfarb–Shanno (BFGS) and its limited-memory version algorithm (L-BFGS), which determine the descent direction by preconditioning the gradient with curvature information.

3.1.4 Sensitivity matrix

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

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial u} \frac{\partial u}{\partial \theta} + \frac{\partial L}{\partial \theta}. \quad (14)$$

Notice here the distinction between the direct derivative $\frac{d}{d\theta}$ and partial derivative $\frac{\partial}{\partial \theta}$. The two partial derivatives involving the loss function on the right-hand side are usually easy to evaluate. For example, for the loss function in Equation (7), these are simply given by

$$\frac{\partial L}{\partial u} = u - u^{\text{target}}(t_1) \quad \frac{\partial L}{\partial \theta} = 0. \quad (15)$$

In most applications, the empirical component of the loss function $L(\theta)$, that is, the part of the loss that is a function on the data, will depend on θ just through u , meaning $\frac{\partial L}{\partial \theta} = 0$. However, regularization terms added to the loss can directly depend on the parameter θ , that is $\frac{\partial L}{\partial \theta} \neq 0$. In both cases, 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 ,

$$s = \frac{\partial u}{\partial \theta} = \begin{bmatrix} \frac{\partial u_1}{\partial \theta_1} & \cdots & \frac{\partial u_1}{\partial \theta_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_n}{\partial \theta_1} & \cdots & \frac{\partial u_n}{\partial \theta_p} \end{bmatrix} \in \mathbb{R}^{n \times p}. \quad (16)$$

The sensitivity s defined in Equation (16) is a *Jacobian*, that is, a matrix of first derivatives of a vector-valued function.

Notice that the product sv , with $v \in \mathbb{R}^p$, is the directional derivative of the function $u(\theta)$ in the direction v , that is

$$sv = \frac{\partial u}{\partial \theta} v = \lim_{h \rightarrow 0} \frac{u(\theta + hv) - u(\theta)}{h}, \quad (17)$$

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

3.2 Finite differences

Finite differences are arguably the simplest scheme to obtain the derivative of a function. In the case of the function $L : \mathbb{R}^p \mapsto \mathbb{R}$, a first-order Taylor expansion yields to the following expression for the directional derivative

$$\frac{dL}{d\theta_i}(\theta) = \frac{L(\theta + \varepsilon e_i) - L(\theta)}{\varepsilon} + \mathcal{O}(\varepsilon), \quad (18)$$

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

$$\frac{dL}{d\theta_i}(\theta) = \frac{L(\theta + \varepsilon e_i) - L(\theta - \varepsilon e_i)}{2\varepsilon} + \mathcal{O}(\varepsilon^2). \quad (19)$$

While Equation (18) 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).

Finite difference scheme are subject to a number of issues, related to the parameter vector dimension and rounding errors. Firstly, calculating directional derivatives requires at least one extra function evaluations per parameter dimension. For the centered differences approach in Equation (19), this requires a total of $2p$ function evaluations which demands solving the DE each time for a new set of parameters. Second, finite differences involve the subtraction of two closely valued numbers, which can lead to floating point cancellation errors when the step size ε is small (Goldberg 1991). While small values of ε lead to cancellation errors, large values of the stepsize give inaccurate estimations of the derivative. Furthermore, numerical solutions of DEs have errors that are typically larger than machine precision, which leads to inaccurate estimations of the gradient when ε is too small (see also Section 4.1.1). 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).

Despite these caveats, finite differences can prove useful in specific contexts, such as 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

$$Jv \approx \frac{f(u + \varepsilon v, \theta, t) - f(u, \theta, t)}{\varepsilon} \quad (20)$$

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).

3.3 Automatic differentiation

Automatic differentiation (AD) is a technique that generates new code representing derivatives of a given computer program defined by some evaluation procedure. Examples are code representing the tangent linear or adjoint operator of the original parent code (Griewank et al. 2008). The names *algorithmic* and *computational* differentiation are also used in the literature, emphasizing the algorithmic rather than automatic nature of AD (Griewank et al. 2008; Margossian 2019; Naumann 2011). Any computer program implementing a given function 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 their capacity to differentiate complex programs that include control flow, such as branching, loops or recursions.

AD falls under the category of discrete methods. Depending on whether the concatenation of the elementary derivatives is done as the program is executed (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* mode AD, respectively. Neither forward nor reverse mode is more efficient in all cases (Griewank 1989), as we will discuss in Section 3.3.3.

3.3.1 Forward mode

Forward mode AD can be implemented in different ways depending on the data structures we use when representing a computer program. Examples of these data structures include dual numbers and computational graphs (Baydin et al. 2017). These representations are mathematically equivalent and lead to the same implementation except for details in the compiler optimizations with respect to floating point ordering.

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 numbers, we can represent dual numbers as an ordered pair (x_1, x_2) , sometimes known as Argand pair, or in the rectangular form

$$x_\epsilon = x_1 + \epsilon x_2, \quad (21)$$

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 allows 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

$$x_\epsilon + y_\epsilon = (x_1 + y_1) + \epsilon(x_2 + y_2) \quad x_\epsilon y_\epsilon = x_1 y_1 + \epsilon(x_1 y_2 + x_2 y_1). \quad (22)$$

From these last examples, we can see that the derivative component of the dual number carries the information of the derivatives when combining operations (e.g., when 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, as evident in how the output of any scalar functions is extended to a dual number output:

$$\begin{aligned} f(x_1 + \epsilon x_2) &= f(x_1) + \epsilon x_2 f'(x_1) + \epsilon^2 \cdot (\dots) \\ &= f(x_1) + \epsilon x_2 f'(x_1). \end{aligned} \quad (23)$$

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 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 . 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).

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 directed acyclic graph (DAG) 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 $v_m = L(\theta)$ 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 the other, a quantity known as the tangent, using the Bauer formula (Bauer 1974; Oktay et al. 2020)

$$\frac{\partial v_j}{\partial v_i} = \sum_{\substack{\text{paths } w_0 \rightarrow w_1 \rightarrow \dots \rightarrow w_K \\ \text{with } w_0 = v_i, w_K = v_j}} \prod_{k=0}^{K-1} \frac{\partial w_{k+1}}{\partial w_k}, \quad (24)$$

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 $j = -p + 1, -p + 1, \dots, m$ using the recursion

$$\frac{\partial v_j}{\partial v_i} = \sum_{w \text{ such that } w \rightarrow v_j} \frac{\partial v_j}{\partial w} \frac{\partial w}{\partial v_i} \quad (25)$$

Since every variable node w such that $w \rightarrow v_j$ is an edge of the computational graph has an index less than j , we can iterate this procedure as we run the computer program and solve for both the function and its derivative. 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 (25).

3.3.2 Reverse mode

Reverse mode AD is also known as the adjoint, or cotangent linear mode, or backpropagation in the field of machine learning. The reverse mode of AD 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 on the number of parameters to differentiate (Griewank 1989; Wolfe 1982), a point known as the *cheap gradient principle* (Griewank 2012). Given a DAG of operations defined by a Wengert list, we can compute gradients of any given function in the same fashion as Equation (25) but in decreasing order $j = m, m - 1, \dots, -p + 1$ as

$$\bar{v}_i = \frac{\partial \ell}{\partial v_i} = \sum_{w \text{ such that } v_i \rightarrow w} \bar{w} \frac{\partial w}{\partial v_i}. \quad (26)$$

In this context, the notation $\bar{w} = \frac{\partial L}{\partial w}$ 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 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 4.1.2.3).

3.3.3 AD connection with JVPs and VJPs

Forward and reverse AD is based on the sequential evaluation of Jacobian-vector products (JVPs) and vector-Jacobian products (VJPs), respectively. Let us consider for example the case of a loss function $L : \mathbb{R}^p \mapsto \mathbb{R}$ taking a total of p arguments as inputs that is computed using the evaluation procedure $L(\theta) = \ell \circ g_k \circ \dots \circ g_2 \circ g_1(\theta)$, 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. If we perturb the parameter $\theta \rightarrow \theta + \delta\theta$, this will produce a perturbation $L(\theta) \rightarrow L(\theta) + \delta L$ in the loss function that can be computed at first order in $\delta\theta$ using the chain rule as:

$$\delta L = \nabla_\theta L \cdot \delta\theta = \nabla \ell \cdot Dg_k \cdot Dg_{k-1} \cdot \dots \cdot Dg_2 \cdot Dg_1 \cdot \delta\theta, \quad (27)$$

with Dg_i the Jacobian of each intermediate function evaluated at the intermediate values $g_{i-1} \circ g_{i-2} \circ \dots \circ g_i(\theta)$ (Giering et al. 1998).

In forward AD, we can compute δL from Equation (27) by defining the intermediate perturbation δg_j as the sequential evaluation of the JVP given by the map between tangent spaces $\delta x \mapsto Dg_j(x) \cdot \delta x$ (Griewank et al. 2008):

$$\delta g_0 = \delta\theta \quad (28)$$

$$\delta g_j = Dg_j \cdot \delta g_{j-1} \quad j = 1, 2, \dots, k \quad (29)$$

$$\delta L = \nabla \ell \cdot \delta g_k. \quad (30)$$

For $\|\delta\theta\|_2 = 1$, this procedure will return δL as the value of the directional derivative of L evaluated at θ in the direction $\delta\theta$ (see Equation (17)). In order to compute the full gradient $\nabla L \in \mathbb{R}^p$, we need to perform this operation $O(p)$ times, which requires a total of $p(d_2d_1 + d_3d_2 + \dots + d_kd_{k-1} + d_k) = O(kp)$ operations.

In the case of reverse AD, we observe that $\nabla \ell \in \mathbb{R}^{d_k}$ is a vector, so we can instead compute δL for all possible perturbations $\delta\theta$ by solving the multiplication involved in Equation (27) starting from the left-hand side. This is carried by the sequential definition of intermediate variables \bar{g}_j computed as VJPs that map between co-tangent (or normal spaces) $\bar{y} \mapsto \bar{y}^T \cdot Dg_j$:

$$\bar{g}_k^T = \nabla \ell \quad (31)$$

$$\bar{g}_{j-1}^T = \bar{g}_j^T \cdot Dg_j \quad j = k, k-1, \dots, 1 \quad (32)$$

$$\nabla L = \bar{g}_0. \quad (33)$$

Since this procedure needs to be evaluated just once to evaluate ∇L , we conclude that reverse AD requires a total of $d_kd_{k-1} + d_{k-1}d_{k-2} + \dots + d_2d_1 + d_1p = O(k+p)$ operations.

The reverse mode will in general be faster when $1 \ll p$. This example is illustrated in Figure 2. 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 $O(pk + q)$ and the cost of reverse is $O(p + kq)$. When the function to differentiate has a larger input space than output ($q \ll p$), AD in reverse mode is

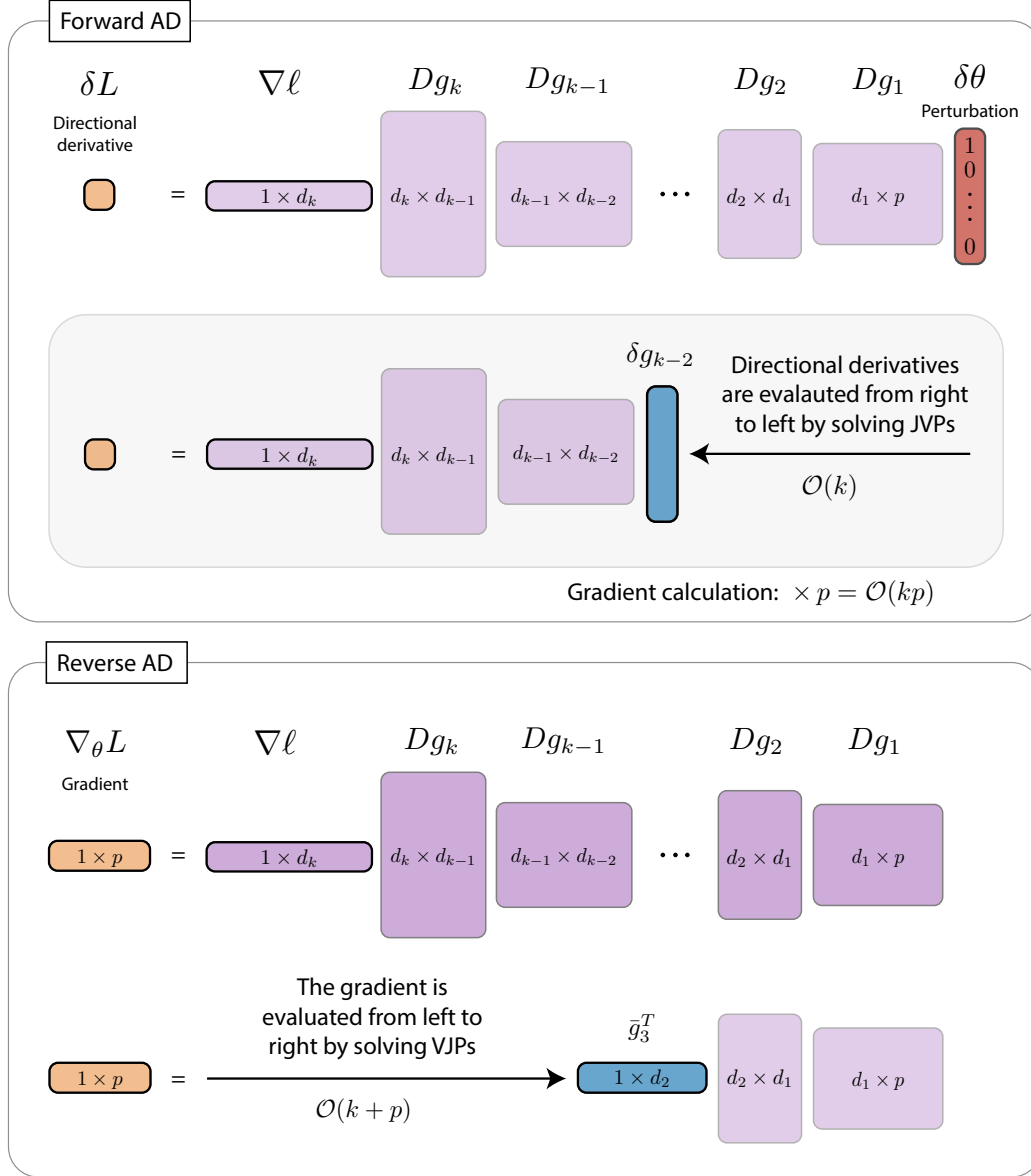


Figure 2: Comparison between forward and reverse AD. Changing the order of Jacobian and vector multiplications changes the total number of floating-point operations, which leads to different computational complexities between forward and reverse mode. When computing directional derivatives with forward AD, there is a total of $\mathcal{O}(k)$ JVPs that need to be computed, which considering we need to repeat this procedure p times gives a total complexity of $\mathcal{O}(kp)$. This is the opposite of what happens when we carry the VJPs 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 $\mathcal{O}(k + p)$.

more efficient as it propagates the chain rule by computing VJPs. For this reason, reverse AD is often preferred in both modern machine learning and inverse methods. However, notice that reverse mode AD requires saving 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 2019).

In practice, most AD systems are reduced to the computation of only directional derivatives (JVPs) or gradients (VJPs) (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 $J e_i$, 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 $e_j \in \mathbb{R}^n$. In other words, forward AD computes Jacobians column-by-column while reverse AD does it row-by-row.

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 newer 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

$$\frac{dL}{d\theta} = \lim_{\varepsilon \rightarrow 0} \frac{\text{Im}(L(\theta + i\varepsilon))}{\varepsilon}, \quad (34)$$

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

$$L(\theta + i\varepsilon) = L(\theta) + i\varepsilon \frac{dL}{d\theta} - \frac{1}{2}\varepsilon^2 \frac{d^2L}{d\theta^2} + \mathcal{O}(\varepsilon^3). \quad (35)$$

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

$$\frac{dL}{d\theta} = \frac{\text{Im}(L(\theta + i\varepsilon))}{\varepsilon} + \mathcal{O}(\varepsilon^2). \quad (36)$$

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 applicable to locally complex analytical functions (Martins et al. 2003) and does not outperform AD (see Sections 4.1 and 6). One additional limitation is that it requires the evaluation of mathematical functions with small complex values, e.g., operations such as $\sin(1 + 10^{-16}i)$, which are not necessarily always computable to high accuracy with modern math libraries. Extension to higher order derivatives can be obtained by introducing multicomplex variables (Lantone et al. 2012).

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 assign variable names, expressions, operations, and literals to *algebraic* objects. For example, the relation $y = x^2$ is interpreted as expression with two variables, x and y , and the symbolic system generates the derivative $y' = 2 \times x$ with 2 a numeric literal, \times a binary operation, and x the same variable assignment as in the original expression.

The general issue with symbolic differentiation is *expression swell*, i.e. the fact that the size of a derivative expression can be much larger than the original expression (Baydin et al. 2017). One way to visualize this swell is to note that the product rule grows and expression of $f(x)g(x)$ into two expressions, namely $\frac{d}{dx}(f(x)g(x)) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx}$, and thus the composition of many functions leads to a large derivative expression. AD avoids expression swell by instead numerically calculating the derivative of a given expression at some fixed value, never representing the general derivative but only at the values obtained by the forward pass. This eager evaluation of the derivative around a given value forces the intermediate computation into the JVPs or VJP form as a way to continually pass forward/reverse the current state. Meanwhile, symbolic differentiation can represent the complete derivative expression and thus avoid being forced into a given computation order, but at the memory cost of having to represent larger expressions.

However, it is important to acknowledge the close relationship between AD and symbolic differentiation. AD uses symbolic differentiation in its definition of primitives which are then chained together in a specific way to form VJPs and vector products. Forward AD can be expressed as a form of symbolic differentiation with a specific choice of common subexpression elimination, i.e. forward AD can be expressed as a symbolic differentiation with a specific choice of how to accumulate the intermediate calculations so that expression growth can be avoided (Dürbaum et al. 2002; Elliott 2018; Juedes 1991; Laue 2019). However, general symbolic differentiation can have many other choices for the differentiation order, and does not in general require computation using the JVPs or VJPs (Baydin et al. 2017). This is apparent for example when computing sparse Jacobians, where generally symbolic differentiation computes entries element-by-element while forward AD computes the matrix column-by-column and reverse AD computes row-by-row (see Section 4.1.2.5).

3.6 Forward sensitivity equations

An easy way to derive an expression for the sensitivity s defined in Equation (16) is by deriving the forward sensitivity equations (Ramsay et al. 2017), a method also referred to as continuous local sensitivity analysis (CSA). If we consider the original ODE given by Equation (1) and we differentiate with respect to θ , we then obtain

$$\frac{d}{d\theta} \left(\frac{du}{dt} - f(u(t; \theta), \theta, t) \right) = 0. \quad (37)$$

Assuming that a 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

$$\frac{d}{d\theta} \frac{du}{dt} = \frac{d}{d\theta} f(u(t; \theta), \theta, t) = \frac{\partial f}{\partial \theta} + \frac{\partial f}{\partial u} \frac{du}{d\theta}. \quad (38)$$

Identifying the sensitivity matrix $s(t)$ now as a function of time, we obtain the *sensitivity differential equation*

$$\frac{ds}{dt} = \frac{\partial f}{\partial u} s + \frac{\partial f}{\partial \theta}. \quad (39)$$

The initial condition is simply given by $s(t_0) = \frac{du_0}{d\theta}$, which is zero unless the initial condition explicitly depends on the parameter θ . Both the original ODE of size n and the forward sensitivity equation of size np are solved simultaneously, which is necessary since the forward sensitivity DE directly depends on the value of $u(t; \theta)$. This implies that as we solve the ODE, 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 forward sensitivity equations find the derivative 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 $u(t; \theta)$ and $s(t)$ as we solve the DE forward in time, without the need of backtracking any operation through the solver. By solving the forward sensitivity equation and the original ODE for $u(t; \theta)$ simultaneously, we ensure that by the end of the forward step we have calculated both $u(t; \theta)$ and $s(t)$.

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. 2000) or in ocean state estimation (Wunsch 2008; Wunsch et al. 2007).

The idea of the adjoint method is to treat the DE 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. 2000). 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).

3.7.1 Adjoint state equations

The derivation of the discrete adjoint equations is carried out once the numerical scheme for solving Equation (1) has been specified. Given a discrete sequence of timesteps t_0, t_1, \dots, t_M , we aim to find approximate numerical solutions $u^m \approx u(t_m; \theta)$. Any numerical solver, including the ones discussed in Section 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 $U = (u_1, u_2, \dots, u_M) \in \mathbb{R}^{nM}$, and we have combined the system of equations defined by the iterative solver as $G(U; \theta) = (g_1(u^1; \theta), \dots, g_M(u^M; \theta)) = 0$ (see Equation (4)).

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

$$L(U, \theta) = \frac{1}{2} \sum_{m=1}^M w_m \|u^m - u_m^{\text{obs}}\|_2^2, \quad (40)$$

with u_m^{obs} the observed time-series, and $w_m \geq 0$ some arbitrary weights (potentially, many of them equal to zero). Similarly to Equation (14) we have

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} + \frac{\partial L}{\partial U} \frac{\partial U}{\partial \theta}. \quad (41)$$

By differentiating the constraint $G(U; \theta) = 0$, we obtain

$$\frac{dG}{d\theta} = \frac{\partial G}{\partial \theta} + \frac{\partial G}{\partial U} \frac{\partial U}{\partial \theta} = 0, \quad (42)$$

which is equivalent to

$$\frac{\partial U}{\partial \theta} = - \left(\frac{\partial G}{\partial U} \right)^{-1} \frac{\partial G}{\partial \theta}. \quad (43)$$

Replacing this last expression into Equation (41), we obtain

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \frac{\partial L}{\partial U} \left(\frac{\partial G}{\partial U} \right)^{-1} \frac{\partial G}{\partial \theta}. \quad (44)$$

The important trick used in the discrete adjoint method is the rearrangement of the multiplicative terms involved in equation (44). Computing the full Jacobian/sensitivity $\partial U / \partial \theta$ will be computationally expensive and involves the product of two matrices (Equation (43)). 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 on the intermediate variable $G(U; \theta)$, we can make the same computation more efficient. This leads to the definition of the adjoint $\lambda \in \mathbb{R}^{nM}$ as the solution of the linear system of equations

$$\left(\frac{\partial G}{\partial U} \right)^T \lambda = \left(\frac{\partial L}{\partial U} \right)^T, \quad (45)$$

or equivalently,

$$\lambda^T = \frac{\partial L}{\partial U} \left(\frac{\partial G}{\partial U} \right)^{-1}. \quad (46)$$

Replacing Equation (46) into (44) yields

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \lambda^T \frac{\partial G}{\partial \theta}. \quad (47)$$

These ideas are summarized in the diagram in Figure 3, where we can also see an 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 (45) is a linear system of equations, even when the original system $G(U; \theta) = 0$ is not necessarily linear in U . This means that while solving the original ODE may require multiple iterations in order to solve the non-linear system $G(U; \theta) = 0$ (e.g., by using Krylov methods), the backwards step to compute the adjoint is one single linear system of equations.

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 numerical solver, where at every step we need to solve the equation $u^{m+1} = g_m(u^m; \theta) = A_m(\theta) u^m + b_m(\theta)$, where $A_m(\theta) \in \mathbb{R}^{n \times n}$ and $b_m(\theta) \in \mathbb{R}^n$ are defined by the numerical solver (Johnson 2012). This condition can be written in a more compact manner as $G(U; \theta) = A(\theta)U - b(\theta) = 0$, that is

$$A(\theta)U = \begin{bmatrix} \mathbb{I}_n & 0 & & & \\ -A_1 & \mathbb{I}_n & 0 & & \\ & -A_2 & \mathbb{I}_n & 0 & \\ & & & \ddots & \\ & & & & -A_{M-1} & \mathbb{I}_n \end{bmatrix} \begin{bmatrix} u^1 \\ u^2 \\ u^3 \\ \vdots \\ u^M \end{bmatrix} = \begin{bmatrix} A_0 u_0 + b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{M-1} \end{bmatrix} = b(\theta), \quad (48)$$

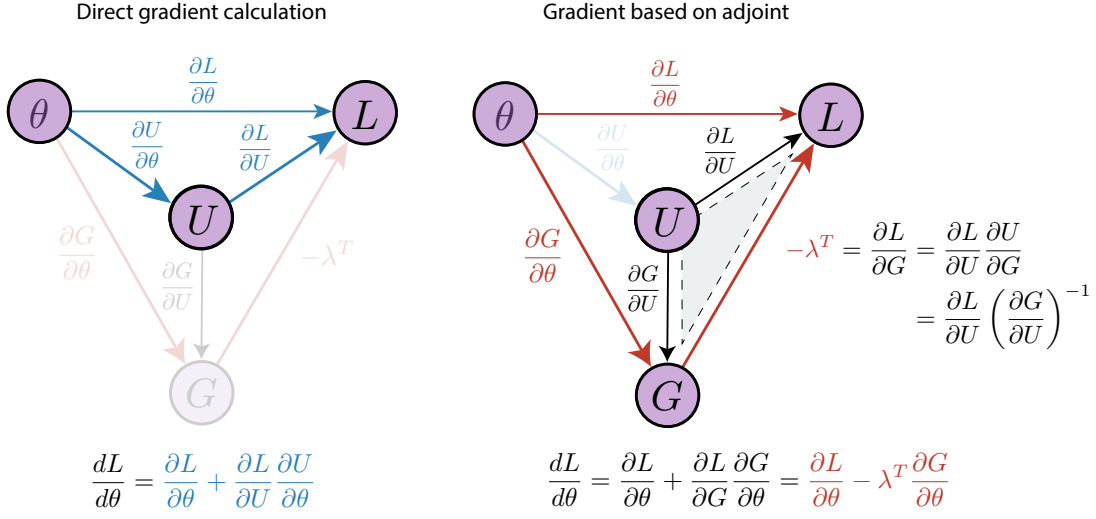


Figure 3: Diagram showing how gradients are computed using discrete adjoints. On the left, we see how gradients will be computed if we use the chain rule applied to the directed triangle defined by the variables θ , U , and L (blue arrows). However, we can define the intermediate vector variable $G = G(U; \theta)$, 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 $G(U; \theta) = 0$.

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

$$\frac{\partial G}{\partial \theta} = \frac{\partial A}{\partial \theta} U - \frac{\partial b}{\partial \theta}, \quad (49)$$

so the desired gradient in Equation (47) can be computed as

$$\frac{dL}{d\theta} = \frac{\partial L}{\partial \theta} - \lambda^T \left(\frac{\partial A}{\partial \theta} U - \frac{\partial b}{\partial \theta} \right) \quad (50)$$

with λ the discrete adjoint obtained by solving the linear system in Equation (45),

$$A(\theta)^T \lambda = \begin{bmatrix} \mathbb{I}_n & -A_1^T & & & \\ 0 & \mathbb{I}_n & -A_2^T & & \\ & 0 & \mathbb{I}_n & -A_3^T & \\ & & & \ddots & -A_{M-1}^T \\ & & & 0 & \mathbb{I}_n \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \vdots \\ \lambda_M \end{bmatrix} = \begin{bmatrix} w_1(u^1 - u_1^{\text{obs}}) \\ w_2(u^2 - u_2^{\text{obs}}) \\ w_3(u^3 - u_3^{\text{obs}}) \\ \vdots \\ w_M(u^M - u_M^{\text{obs}}) \end{bmatrix} = \frac{\partial L}{\partial U}^T. \quad (51)$$

This is a linear system of equations with the same size as 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_m is by noticing that the system of equations (51) is equivalent to the final value problem

$$\lambda_m = A_m^T \lambda_{m+1} + w_m(u^m - u_m^{\text{obs}}) \quad (52)$$

with final condition λ_M . This means that we can efficiently compute the adjoint equation in reverse mode, starting from the final state λ_M and computing the values of λ_m in decreasing index order. Unless the loss function L is linear in U , this procedure requires knowledge of the value of u^m (or some equivalent form of it) at any given timestep t_m .

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 DEs for the adjoint variable and using this to compute the gradient in a more efficient manner. The continuous adjoint method follows the same logic as the discrete adjoint method, but where the discretization of the DE does not happen until the very last step, when the solutions need to be solved numerically.

Consider an integrated loss function defined in Equation (10) of the form

$$L(u; \theta) = \int_{t_0}^{t_1} h(u(t; \theta), \theta) dt \quad (53)$$

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

$$\frac{dL}{d\theta} = \int_{t_0}^{t_1} \left(\frac{\partial h}{\partial \theta} + \frac{\partial h}{\partial u} s(t) \right) dt. \quad (54)$$

Just as in the case of the discrete adjoint method, the complicated term to evaluate in the last expression is the sensitivity $s(t)$. Again, the trick is to evaluate the VJP $\frac{\partial h}{\partial u} s(t)$ by first defining an intermediate adjoint variable. The continuous adjoint equation now is obtained by finding the dual/adjoint equation associated to the forward sensitivity equation using the weak formulation of Equation (39) (Brézis 2011). The adjoint equation is obtained by writing the forward sensitivity equation in the form

$$\int_{t_0}^{t_1} \lambda(t)^T \left(\frac{ds}{dt} - \frac{\partial f}{\partial u} s - \frac{\partial f}{\partial \theta} \right) dt = 0, \quad (55)$$

where this equation must be satisfied for every suitable function $\lambda : [t_0, t_1] \mapsto \mathbb{R}^n$ in order for Equation (39) to be true. The next step is to get rid of the time derivative applied to the sensitivity $s(t)$ using integration by parts:

$$\int_{t_0}^{t_1} \lambda(t)^T \frac{ds}{dt} dt = \lambda(t_1)^T s(t_1) - \lambda(t_0)^T s(t_0) - \int_{t_0}^{t_1} \frac{d\lambda^T}{dt} s(t) dt. \quad (56)$$

Replacing this last expression into Equation (55) we obtain

$$\int_{t_0}^{t_1} \left(-\frac{d\lambda^T}{dt} - \lambda(t)^T \frac{\partial f}{\partial u} \right) s(t) dt = \int_{t_0}^{t_1} \lambda(t)^T \frac{\partial f}{\partial \theta} dt - \lambda(t_1)^T s(t_1) + \lambda(t_0)^T s(t_0). \quad (57)$$

At first glance, there is nothing particularly interesting about this last equation. However, both Equations (54) and (57) involve $s(t)$ in a VJP. Since Equation (57) must hold for every function $\lambda(t)$,

we can pick $\lambda(t)$ to make the terms involving $s(t)$ in Equations (54) and (57) to perfectly coincide. This is done by defining the adjoint $\lambda(t)$ to be the solution of the new system of differential equations

$$\frac{d\lambda}{dt} = -\frac{\partial f^T}{\partial u} \lambda - \frac{\partial h^T}{\partial u} \quad \lambda(t_1) = 0. \quad (58)$$

Notice that the adjoint equation is defined with the final condition at t_1 , meaning that it needs to be solved backwards in time from t_1 to t_0 . The definition of the adjoint $\lambda(t)$ as the solution of this last ODE simplifies Equation (57) to

$$\int_{t_0}^{t_1} \frac{\partial h}{\partial u} s(t) dt = \lambda(t_0)^T s(t_0) + \int_{t_0}^{t_1} \lambda(t)^T \frac{\partial f}{\partial \theta} dt. \quad (59)$$

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

$$\frac{dL}{d\theta} = \lambda(t_0)^T s(t_0) + \int_{t_0}^{t_1} \left(\frac{\partial h}{\partial \theta} + \lambda^T \frac{\partial f}{\partial \theta} \right) dt \quad (60)$$

The full algorithm to compute the full gradient $\frac{dL}{d\theta}$ can be described as follows: (i) Solve the original ODE given by $\frac{du}{dt} = f(u, t, \theta)$, (ii) Solve the reverse adjoint ODE given by Equation (58), (iii) Compute the gradient using Equation (60).

3.9 Mathematical comparison of the methods

In Sections 3.2 – 3.8 we focused on merely introducing each one of the sensitivity methods classified in Figure 1 as separate methods, postponing the discussion about their points in common. In this section, we compare one-to-one these methods and highlight differences and parallels between them.

3.9.1 Forward AD and complex step differentiation

Both AD based on dual numbers 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, AD gives the exact gradient value, whereas complex step differentiation relies on numerical approximations that are valid only when the stepsize ε is small. In Figure 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 $\varepsilon \rightarrow 0$ 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. Further notice 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).

3.9.2 Discrete adjoints and reverse AD

Both discrete adjoint methods and reverse AD are classified as discrete and reverse methods (see Figure 1). Furthermore, both methods introduce an intermediate adjoint associated with the partial derivative of the loss function (output variable) with respect to intermediate variables of the forward computation. In the case of reverse AD, this adjoint is defined with the notation \bar{w} (Equation (26)),

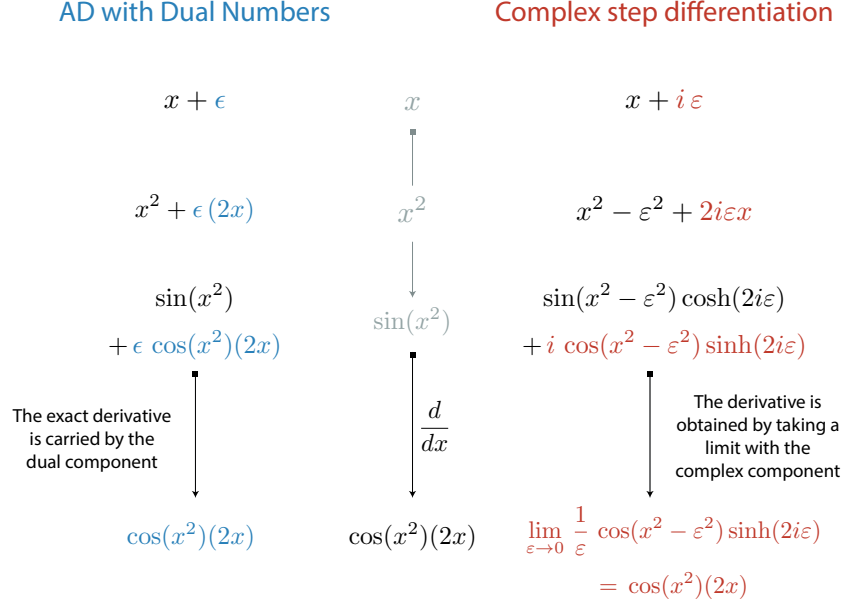


Figure 4: Comparison between AD implemented with dual numbers and complex step differentiation. For the simple case of the function $f(x) = \sin(x^2)$, we can see how each operation is carried in the forward step by the dual component (blue) and the complex component (red). Whereas AD gives the exact 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.

while in the discrete adjoint method this correspond to each one of the variables $\lambda_1, \lambda_2, \dots, \lambda_M$ (Equation (51)). In this section we show that both methods are mathematically equivalent (Li et al. 2020a; Zhu et al. 2021), 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^m satisfy the equation $u^{m+1} = g_m(u^m; \theta)$. Following the same schematics as in Figure 3, we represent the computational graph of the numerical method using the intermediate variables g_1, g_2, \dots, g_{M-1} . The dual/adjoint variables defined in reverse AD in this computational graph are given by

$$\bar{g}_m^T = (\bar{u}^{m+1})^T \frac{\partial u^{m+1}}{\partial g_m} = (\bar{g}_{m+1})^T \frac{\partial g_{m+1}}{\partial u^{m+1}} + \left(\frac{\partial L}{\partial u^{m+1}} \right)^T. \quad (61)$$

The updates of \bar{g}_m then mathematically coincide with the updates in reverse mode of the adjoint variable λ_m (see Equation (52)) mapping between tangent spaces (see Section 3.3.3).

Modern numerical solvers use functions g_m that correspond to nested functions, meaning $g_m = g_m^{(k_m)} \circ g_m^{(k_m-1)} \circ \dots \circ g_m^{(1)}$. This is certainly the case for implicit methods when u^m is computed as the solution of $g_m(u^m; \theta) = 0$ using an iterative Newton 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 2019). A solution to this problem is to introduce a customized *super node* that directly encapsulates the

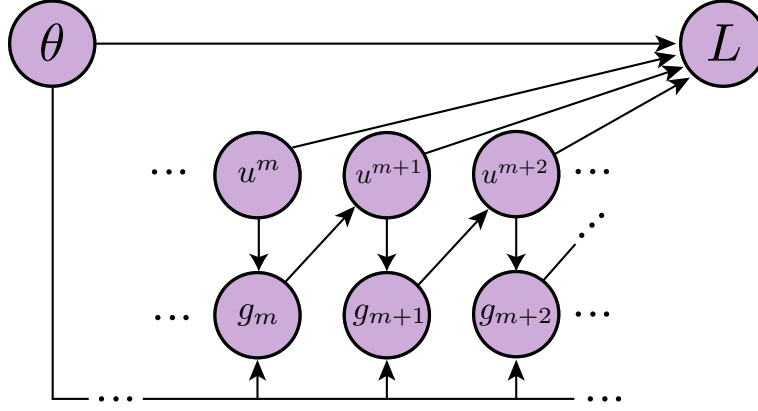


Figure 5: Computational graph associated to the discrete adjoint method. Reverse AD applied on top of the computational graph leads to the update rules for the discrete adjoint. The adjoint variable λ_i in the discrete adjoint method coincides with the adjoint variable \bar{g}_i defined in the backpropagation step.

contribution to the full adjoint in \bar{g}_m without computing the adjoint for each intermediate function $g_m^{(j)}$. Provided with the value of the Jacobian matrices $\frac{\partial g_m}{\partial u^m}$ and $\frac{\partial g_m}{\partial \theta}$, we can use the implicit function theorem to find $\frac{\partial u^m}{\partial \theta}$ as the solution of the linear system of equations

$$\frac{\partial g_m}{\partial u^m} \frac{\partial u^m}{\partial \theta} = -\frac{\partial g_m}{\partial \theta} \quad (62)$$

and implement AD by directly solving this new system of equations (Bell et al. 2008; Christianson 1994, 1998). 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).

3.9.3 Consistency: forward AD and forward sensitivity equations

The forward sensitivity equations can also be solved in discrete forward mode by numerically discretizing the original ODE and later deriving the discrete forward sensitivity equations (Ma et al. 2021). For most cases, this leads to the same result as 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

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{bmatrix} \rightarrow \begin{bmatrix} \theta_1 + \epsilon_1 \\ \theta_2 + \epsilon_2 \\ \vdots \\ \theta_p + \epsilon_p \end{bmatrix} \quad (63)$$

where $\epsilon_i \epsilon_j = 0$ for all pairs of i and j (see Section 3.3.1.1). The dependency of the solution u of the ODE on the parameter θ is now expanded following Equation (23) as

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \rightarrow \begin{bmatrix} u_1 + \sum_{j=1}^p \frac{\partial u_1}{\partial \theta_j} \epsilon_j \\ u_2 + \sum_{j=1}^p \frac{\partial u_2}{\partial \theta_j} \epsilon_j \\ \vdots \\ u_n + \sum_{j=1}^p \frac{\partial u_n}{\partial \theta_j} \epsilon_j \end{bmatrix} = u + s \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_p \end{bmatrix}, \quad (64)$$

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 forward sensitivity equation (Equation (39)). For example, for the forward Euler method this gives

$$\begin{aligned} u^{m+1} + s^{m+1} \epsilon &= u^m + s^m \epsilon + \Delta t_m f(u^m + s^m \epsilon, \theta + \epsilon, t_m) \\ &= u^m + \Delta t_m f(u^m, \theta, t_m) + \left[s^m + \Delta t_m \left(\frac{\partial f}{\partial u} s^m + \frac{\partial f}{\partial \theta} \right) \right] \epsilon. \end{aligned} \quad (65)$$

The dual component corresponds to the forward Euler discretization of the forward sensitivity equation, with s^m 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) gives the following identities

$$u^{m+1} + s^{m+1} \epsilon = u^m + s^m \epsilon + \Delta t_m \sum_{i=1}^s b_i (k_i + \dot{k}_i \epsilon) \quad (66)$$

$$k_i + \dot{k}_i \epsilon = f \left(u^m + \sum_{j=1}^s a_{ij} k_j + \left(s^m + \sum_{j=1}^s a_{ij} \dot{k}_j \right) \epsilon, \theta + \epsilon, t_m + c_i \Delta t_m \right) \quad (67)$$

with \dot{k}_i the dual variable associated to k_i . The partial component in Equation (67) carrying the coefficient ϵ gives

$$\begin{aligned} \dot{k}_i &= \frac{\partial f}{\partial u} \left(u^m + \sum_{j=1}^s a_{ij} k_j, \theta, t_m + c_i \Delta t_m \right) \left(s^m + \sum_{j=1}^s a_{ij} \dot{k}_j \right) \\ &\quad + \frac{\partial f}{\partial \theta} \left(u^m + \sum_{j=1}^s a_{ij} k_j, \theta, t_m + c_i \Delta t_m \right), \end{aligned} \quad (68)$$

which coincides with the Runge-Kutta scheme we would obtain for the original forward 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.

Note that consistency does not imply that an ODE solver is necessarily correct or stable under such a transformation. Consistency of the adjoint may involve other aspects of the solver, such as adaptivity, error control, and the choice of the discretization scheme. A common case where continuous methods may fail is when the discretization step is applied without controlling for the joint error of the solution of the DE and its sensitivity (Gunzburger 2002). In Section 4.1.2.4, we demonstrate that common implementations of adaptive ODE solvers may not compute the right gradient when forward AD is applied to solver even though the process is mathematically consistent. This highlights that additional factors beyond consistency must be considered when investigating whether an implementation is convergent.

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 analytically,

without a priori consideration of the numerical scheme used to solve it. In some sense, we can think of the discrete adjoint $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_M)$ in Equation (51) 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. In general, discrete and continuous adjoints will lead to different numerical solutions for the sensitivity, meaning that the discretization and differentiation step do not commute (Gunzburger 2002; Jensen et al. 2014; Nadarajah et al. 2000). However, as the error of the numerical solver decreases, we further expect the discrete and continuous adjoint to lead to the same correct solution. Furthermore, since the continuous adjoint method requires to numerically solve the adjoint, we are interested in the relative accuracy of the forward and reverse 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) satisfy the condition $b_i \neq 0$ for all $i = 1, 2, \dots, s$, then the discrete adjoint is a consistent estimate of the continuous adjoint with same level of convergence as for 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 backward 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 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).

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 3, i.e., between those that apply direct differentiation at the algorithmic level and those that are based on numerical solvers. The former require a much different implementation since they are agnostic with respect to the mathematical and numerical properties of the ODE. The latter family of methods that are based on numerical solvers include the forward sensitivity equations and the adjoint methods. This section is then divided in two parts:

- **Direct methods.** (Section 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
 - ▷ Forward sensitivity equations (Section 4.2.1)
 - ▷ Discrete and continuous adjoint methods (Section 4.2.2)

While these methods can be implemented in different programming languages, here we consider examples based on the Julia programming language. Julia is a recent but mature programming language that has already a large tradition in implementing packages aiming to advance DP (Bezanson et al. 2017, 2012), which a strong emphasis on DE solvers (Rackauckas et al. 2020, 2016). Nevertheless, in reviewing existing work, we also point to applications developed in other programming languages.

The GitHub repository <https://github.com/ODINN-SciML/DiffEqSensitivity-Review> contains both text and code used in this manuscript. See Appendix A for a complete description of the scripts provided. We use the symbol ♣ to reference code.

4.1 Direct methods

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

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 directional derivative. In Julia, these methods are implemented in `FiniteDiff.jl` and `FiniteDifferences.jl`, which already include subroutines to determine optimal step-sizes. However, finite differences are less accurate and as costly as forward AD (Griewank 1989) and complex-step differentiation. Figure 6 illustrates the error in computing the derivative 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. Here we consider the solution of the simple harmonic oscillator $u'' + \theta^2 u = 0$ with initial condition $u(0) = 0$ and $u'(0) = 1$, which has analytical solution $u_\theta^{\text{true}}(t) = \sin(\theta t)/\theta$. The numerical solution $u_\theta^{\text{num}}(t)$ can be obtained by solving the following ODE:

$$\begin{cases} \frac{du_1}{dt} = u_2 & u_1(0) = 0 \\ \frac{du_2}{dt} = -\theta^2 u_1 & u_2(0) = 1. \end{cases} \quad (69)$$

We use $L(\theta) = u_\theta(t_1)$ as our loss function, so that $\frac{dL}{d\theta} = (t_1/\theta) \cos(\theta t_1) - \sin(\theta t_1)/\theta^2$ for $t_1 = 10$. Finite differences are inaccurate for computing the derivative of u_θ^{true} with respect to θ when the stepsize ε is both too small and too large (red dashed line), with a minimum error for $\varepsilon \approx 10^{-6}$. This case is idealistic as u_θ^{true} cannot generally be obtained analytically, so its derivative 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. When the derivative is instead computed using the numerical solution $u_\theta^{\text{num}}(t)$ (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. For this experiment, both relative and absolute tolerances of the numerical solver had been set to 10^{-6} (high tolerance) and 10^{-12} (low tolerance) (see Section 3.1.1).

4.1.2 Automatic differentiation

The AD algorithms described in Section 3.3 can be implemented using different strategies, namely 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 Section 4.1.2.1 we first cover how forward AD is implemented using dual numbers, postponing the discussion about the implementation using computational graphs for reverse AD in Section 4.1.2.2.

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 with a numerical value 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.

```
@kwdef struct DualNumber{F <: AbstractFloat}
```

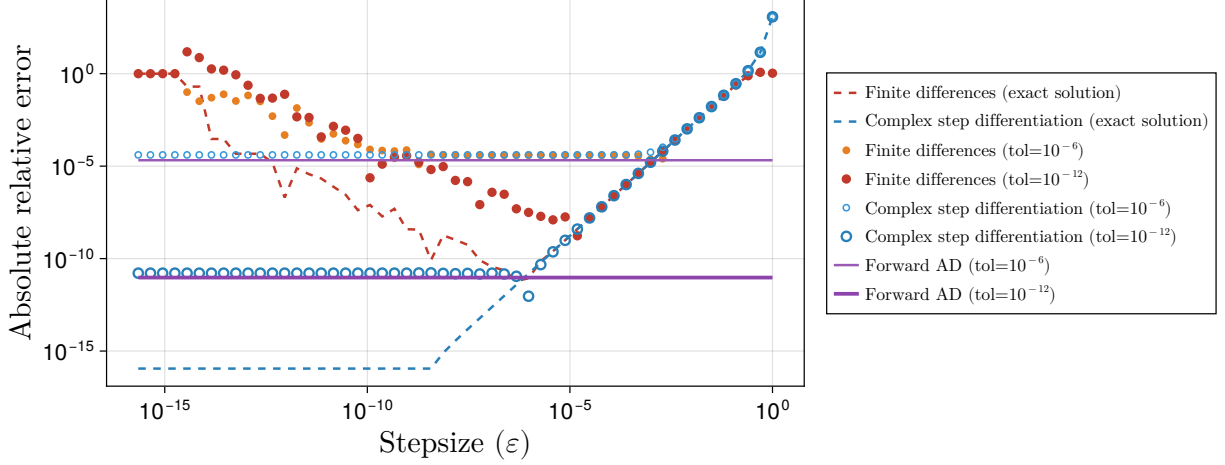


Figure 6: Absolute relative error when computing the gradient of the function $u(t) = \sin(\theta t)/\theta$ with respect to θ at $t = 10.0$ as a function of the stepsize ε for different direct methods. Here, $u(t)$ corresponds to the solution of the differential equation $u'' + \theta^2 u = 0$ with initial condition $u(0) = 0$ and $u'(0) = 1$. The two dashed lines correspond to the case where differentiation is applied on the analytical solution. Dots correspond to the differentiation of the numerically computed solution using the default Tsitouras solver (Tsitouras 2011) from `OrdinaryDiffEq.jl` using different tolerances. Horizontal lines correspond to AD (either forward or reverse). The error when using a numerical solver is larger and it is dependent on the numerical precision of the numerical solver.

♣₁

```

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 (23):

```

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). While `ForwardDiff.jl` defines the interface for defining the tangent of primitive operations, the tangent of different operations are implemented in `DiffRules.jl`. Figure 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 4.1.3) when optimizing by the stepsize ε . 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 derivative calculations (Griewank 1989). This consideration also applies to the other forward methods, including finite differences and complex-step differentiation.

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. The interface for defining primitives is implemented in `ChainRulesCore.jl`, while the primitives themselves are defined in different libraries (eg, `ChainRules.jl`, `SciMLSenstivity.jl`, `NNlib.jl`). 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, \dots, v_m , where $v_i \rightarrow w$ are all edges in the graph, then the pullback $\bar{v}_1, \dots, \bar{v}_m = \mathcal{B}_w(\bar{w})$ 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 multiplication $w = v_1 \times v_2$. Then

$$\bar{v}_1, \bar{v}_2 = v_2 \times \bar{w}, v_1 \times \bar{w} = \mathcal{B}_w(\bar{w}), \quad (70)$$

which is equivalent to using the chain rule as

$$\frac{\partial \ell}{\partial v_1} = \frac{\partial}{\partial v_1}(v_1 \times v_2) \frac{\partial \ell}{\partial w} = v_2 \times \bar{w}, \quad \frac{\partial \ell}{\partial v_2} = v_1 \times \bar{w}. \quad (71)$$

A crucial distinction between AD implementations based on computational graphs is between *static* and *dynamic* methods (Baydin et al. 2017). In the case of a static implementations, 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, as it simplifies performance optimizations to be applied (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. In the case of dynamic methods, these issues are addressed using *tracing* or *tape-based* implementations, where the program structure is transformed into a list of pullback functions that build the graph dynamically at runtime. Popular Julia libraries falling in this category are `Tracker.jl` and `ReverseDiff.jl`. A major drawback of tracing systems is that the pullbacks are constructed with respect to the control flow of the input value and thus do not necessarily generalize to other inputs. This means that the pullback must be reconstructed for each forward pass, limiting the reuse of computational optimizations and inducing higher overhead. Source-to-source AD systems can achieve higher performance by giving a static derivative representation to arbitrary control flow structure, thus allowing for the construction and optimization of pullbacks independent of the input value. These include `Zygote.jl` (Innes et al. 2019), `Enzyme.jl` (Moses et al. 2020; Moses et al. 2021), and `Diffraction.jl`. The existence of these multiple AD packages lead to the development of `AbstractDifferentiation.jl` (Schäfer et al. 2021b) and `DifferentiationInterface.jl` (Dalle et al. 2024), which allows one to combine different methods under the same framework.

4.1.2.3 Discrete checkpointing

In contrast to forward methods, all reverse methods, including backpropagation and adjoint methods, require to access the value of intermediate variables during the propagation of the gradient. For a numerical solver or for time-stepping codes, the number 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 (or the number of time steps). Checkpointing is a technique that can be used for all reverse methods. It 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 reverse mode. Different checkpointing algorithms have been proposed, ranging from static or uniform, multi-level (Giering et al. 1998; Heimbach et al. 2005) to optimized, binomial checkpointing algorithms (Bockhorn et al. 2020; Griewank et al. 2000; Schanen et al. 2023; Walther et al. 2004).

4.1.2.4 When AD is algorithmically correct but numerically wrong

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). In this section we are going to show an example where AD fails when directly applied to an unmodified solution computed with an adaptive stepsize numerical solver (see Section 3.1.1). When performing forward AD through numerical solver, the error used in the stepsize controller needs to naturally account for both the errors induced in the numerical solution of the original ODE and the errors in the dual component carrying the value of the sensitivity. This relation between the numerical solver and AD has been made explicit when we presented the relationship between forward AD and the forward sensitivity equations (Section 3.9.3).

To illustrate this, let us consider the following first-order ODE:

$$\begin{cases} \frac{du_1}{dt} = au_1 - u_1u_2 & u_1(0) = 1 \\ \frac{du_2}{dt} = -au_2 + u_1u_2 & u_2(0) = 1. \end{cases} \quad (72)$$

Notice that for the value of the parameter $a = 1$, this ODE admits an analytical solution $u_1(t) \equiv u_2(t) \equiv 1$, making this problem very simple to solve numerically. The following code solves for the derivative with respect to the parameter a using two different methods. The second method using forward AD with dual numbers declares the `internallnorm` argument for the stepsize controller according to Equation (5) ♣₃.

```
using SciMLSensitivity, OrdinaryDiffEq, Zygote, ForwardDiff

function fiip(du, u, p, t)
    du[1] = p[1] * u[1] - u[1] * u[2]
    du[2] = -p[1] * u[2] + u[1] * u[2]
end

p = [1.]
u0 = [1.0; 1.0]
prob = ODEProblem(fiip, u0, (0.0, 10.0), p);

# Correct gradient computed using
grad0 = Zygote.gradient(p->sum(solve(prob, Tsit5(), u0=u0, p=p, sensealg =
    ForwardSensitivity(), saveat = 0.1, abstol=1e-12, reltol=1e-12)), p)
# grad0 = ([212.71042521681443],)

# Original AD with wrong norm
```

```
grad1 = Zygote.gradient(p->sum(solve(prob, Tsit5(), u0=u0, p=p, sensealg =
    ForwardDiffSensitivity(), saveat = 0.1, internalnorm = (u,t) -> sum(abs2,u/
    length(u)), abstol=1e-12, reltol=1e-12)), p)
# grad1 = ([6278.15677493293],)
```

The reason why the two methods give different answers is because the error estimation by the stepsize controller is ignoring numerical errors in the dual component. In the later case, since the numerical solution of the original ODE is constant, the local estimated error is drastically underestimated to $\text{err}_i^m = 0$, which makes the stepsize Δt_m to increase by a multiplicative factor at every step (see Equations (5) and (6)). This can be fixed by instead considering a norm that accounts for both the primal and dual components in the forward pass,

$$\text{Err}_{\text{scaled}}^m = \left[\frac{1}{n(p+1)} \left(\sum_{i=1}^n \left(\frac{u_i^m - \hat{u}_i^m}{\text{abstol} + \text{reltol} \times \max\{u_i^m, \hat{u}_i^m\}} \right)^2 + \sum_{i=1}^n \sum_{j=1}^p \left(\frac{s_{ij}^m - \hat{s}_{ij}^m}{\text{abstol} + \text{reltol} \times \max\{s_{ij}^m, \hat{s}_{ij}^m\}} \right)^2 \right) \right]^{\frac{1}{2}}, \quad (73)$$

with s^m and \hat{s}^m two different numerical approximations of the sensitivity matrix. This correction now gives the right answer:

```
sse(x::Number) = x^2
sse(x::ForwardDiff.Dual) = sse(ForwardDiff.value(x)) + sum(sse, ForwardDiff.
    partials(x))

totallength(x::Number) = 1
totallength(x::ForwardDiff.Dual) = totallength(ForwardDiff.value(x)) + sum(
    totallength, ForwardDiff.ppartials(x))
totallength(x::AbstractArray) = sum(totallength,x)

grad3 = Zygote.gradient(p->sum(solve(prob, Tsit5(), u0=u0, p=p, sensealg =
    ForwardDiffSensitivity(), saveat = 0.1, internalnorm = (u,t) -> sqrt(sum(x-
    >sse(x),u) / totallength(u)), abstol=abstol, reltol=reltol)), p)
# grad3 = ([212.71042521681392],)
```

This is an example where the form of the numerical solver for the original ODE is affected by the fact we are simultaneously solving for the sensitivity. Notice that current implementations of forward AD inside `SciMLSensitivity.jl` already accounts for this and there is no need to specify the internal norm ♣₃. To highlight the pervasiveness of this issue with respect to AD, we further provide a script with an example in `Difffrax` where the derivative that does not converge to the correct answer as tolerance is decreased to zero ♣₄.

4.1.2.5 Differentiation with sparsity

The total number of computations required to evaluate a gradient/Jacobian using any direct method will depend of its sparsity. When the sparsity pattern is known, colored AD efficiently chunk the calculation of the Jacobian into multiple JVPs or VJP (Gebremedhin et al. 2005). This results in a smaller number of evaluations of JVPs/VJPs compared to the one required to compute all entries of a dense Jacobian (Pal et al. 2024). This can represent a significant advantage for large-scale nonlinear systems and discretized PDEs where sparse Jacobian are commonplace.

Generally AD is recommended as a direct differentiation method due to its generality on programs and the lack of potential memory issues by avoiding expression swell, though there are some

situations where symbolic differentiation may be appropriate. For example, there are certain sparsity patterns by which even colored AD requires computing the full Jacobian via all columns/rows, while computing the sparse Jacobian element-by-element using symbolic differentiation only requires computing the non-zero elements. In other words, depending of the sparsity pattern, symbolic differentiation can be more efficient than AD (Lantoine et al. 2012). An example of this is given by the arrowhead matrix $J_{\text{arrowhead}} \in \mathbb{R}^{n \times n}$ given by:

$$J_{\text{arrowhead}} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \cdots & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 & & 0 & 0 \\ \bullet & 0 & \bullet & 0 & & 0 & 0 \\ \bullet & 0 & 0 & \bullet & & 0 & 0 \\ \vdots & & & & \ddots & & \\ \bullet & 0 & 0 & 0 & & \bullet & 0 \\ \bullet & 0 & 0 & 0 & & 0 & \bullet \end{bmatrix}, \quad (74)$$

where \bullet indicate non trivial zero entries of the Jacobian. In this case, both forward and reverse AD will have to perform n VJPs and JVPs, respectively, and there is no computational benefit of using colored AD. Instead, symbolic differentiation constructs a symbolic representation of the sparse Jacobian entry-by-entry and can fill the Jacobian with $3n - 2$ computations, where each computation is significantly cheaper than each VJP or JVP. Notice that for the arrowhead matrix example, a combination of forward and reverse AD can be used to color the Jacobian with two forward and one reverse AD evaluation.

4.1.3 Complex step differentiation

Modern software already have 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 ODE in Equation (69) to support complex numbers \clubsuit_5 .

```
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 6 further shows the result of performing complex step differentiation using the same example as in Section 4.1.1. We can see from both exact and numerical solution 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 to finite

differences, while for small values of ε the performance of complex step differentiation is slightly worse than AD. This result emphasizes the observation made in Section 3.9.2, namely that complex step differentiation has many aspects in common with finite differences and AD based on dual numbers.

However, the difference between the methods also makes the complex step differentiation sometimes more efficient than both finite differences and AD (Lantoiné 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 4) (Martins et al. 2003). Further notice that complex-step differentiation will work as long as every function involved in the computation is locally analytical. This is a problem with implementations of functions that rely on the absolute function, for example using complex step differentiation on the square function implemented as $f(z) = \text{abs}(z)^2$ will always return zero ($\text{Im}(f(x + i\varepsilon)) = \text{Im}((x + i\varepsilon)(x - i\varepsilon)) = \mathcal{O}(\varepsilon^2)$).

4.2 Solver-based methods

We now move our discussion to DP methods based on numerical solvers. These need to deal with some numerical and computational considerations, including:

- How to handle JVPs and/or VJP
- Stability of the numerical solver, including the original ODE but also the sensitivity/adjoint equations
- Memory-time tradeoff

These factors are further exacerbated by the size n of the ODE and the number p of parameters in the model. Just a few modern scientific software implementations have the capabilities of solving ODE 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 Fortran; SciMLSensitivity.jl in Julia (Rackauckas et al. 2020); Dolfín-adjoint based on the FEniCS Project (Farrell et al. 2013; Mitusch et al. 2019); DENSERKS in Fortran (Alexe et al. 2007); DASPKADJOINT (Cao et al. 2002); and DiffraX (Kidger 2021) and torchdiffeq (Chen 2018) in Python.

4.2.1 Forward sensitivity equation

For systems of equations with few number of parameters, the forward sensitivity equation is useful since the system of $n(p + 1)$ equations composed by Equations (1) and (39) 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 (69) and the forward sensitivity equation can be solved simultaneously using the simple explicit Euler method ♣₆:

```
p = [0.2]
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  $\partial f / \partial p$ 
function  $\partial f \partial p(u, p, t)$ 
    Jac = zeros(length(u), length(p))
    Jac[2,1] = -2*p[1]*u[1]
    return Jac
end

# Jacobian  $\partial f / \partial u$ 
function  $\partial f \partial u(u, p, t)$ 
    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 * ( $\partial f \partial u(u, p, ti)$  * sensitivity +  $\partial f \partial p(u, p, ti)$ )
        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 SciMLSensitivity in the Julia SciML ecosystem, the `ODEForwardSensitivityProblem` method implements continuous sensitivity analysis, which generates the JVPs required as part of the forward sensitivity equations via `ForwardDiff.jl` (see Section 3.9.3) or finite differences. Using SciMLSensitivity reduces the code above to

```

using SciMLSensitivity

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

prob = ODEForwardSensitivityProblem(f!, u0, tspan, p)
sol = solve(prob, Tsit5())

```

For stiff systems of ODEs the use of the forward sensitivity equations can be computationally 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 $\mathcal{O}(n^3 p^3)$. This complexity makes this method expensive for models with large n and/or p unless the solver is able to further specialize on sparsity or properties of the linear solver (i.e. through Newton-Krylov methods).

4.2.1.1 Computing JVPs and VJPs inside the solver

An important consideration is that all solver-based methods have subroutines to compute the JVPs and VJPs involved in the sensitivity and adjoint equations, respectively. 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. In the case of the forward sensitivity equation, this corresponds to the JVPs resulting from the product $\frac{\partial f}{\partial u}s$ in Equation (39). For the adjoint equations, we need to evaluate the term $\lambda^T \frac{\partial f}{\partial \theta}$ for the continuous adjoint method in Equation (60), while for the discrete adjoint method we need to compute the term $\lambda^T \frac{\partial G}{\partial \theta}$ in Equation (46) (which may further coincide with $\lambda^T \frac{\partial f}{\partial \theta}$ for some numerical solvers, but not in the general case). Therefore, the choice of the algorithm to compute JVPs/VJPs can have a significant impact in the overall performance (Schäfer et al. 2021b).

In SUNDIALS, the JVPs/VJPs involved in the sensitivity and adjoint method are handled using finite differences unless specified by the user (Hindmarsh et al. 2005). In FATODE, they can be computed with finite differences, AD, or it can be provided by the user. In the Julia SciML ecosystem, the options `autodiff` and `autojacvec` allow one to customize if JVPs/VJPs are computed using AD, finite differences, or alternatively these are provided by the user. Different AD packages with different performance trade-offs are available for this task (see Section 4.1.2.2), including `ForwardDiff.jl` (Revels et al. 2016), `ReverseDiff.jl`, `Zygote.jl` (Innes et al. 2019), `Enzyme.jl` (Moses et al. 2020), `Tracker.jl`.

4.2.2 Adjoint methods

For complex and large systems (e.g. for $n + p > 50$, as we will discuss in Section 6), 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. Since adjoint methods rely on an additional set of ODEs which are solved, numerical efficiency and stability must further be taken into account at the moment of implementing adjoint methods.

4.2.2.1 Discrete adjoint method

In order to illustrate how the discrete adjoint method can be implemented, the following example shows how to manually solve for the gradient of the solution of (69) using an explicit Euler method

♣₇.

```
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
end
```

	Method	Stability	Non-Stiff Performance	Stiff Performance	Memory
Discrete	ReverseDiffAdjoint	Good	$\mathcal{O}(n + p)$	$\mathcal{O}(n^3 + p)$	High
	TrackerAdjoint	Good	$\mathcal{O}(n + p)$	$\mathcal{O}(n^3 + p)$	High
Continuous	Forward sensitivity eq.	Good	$\mathcal{O}(np)$	$\mathcal{O}(n^3 p^3)$	$\mathcal{O}(1)$
	Backsolve adjoint	Poor	$\mathcal{O}(n + p)$	$\mathcal{O}((n + p)^3)$	$\mathcal{O}(1)$
	Backsolve adjoint ◀	Medium	$\mathcal{O}(n + p)$	$\mathcal{O}((n + p)^3)$	$\mathcal{O}(nK)$
	Interpolating adjoint	Good	$\mathcal{O}(n + p)$	$\mathcal{O}((n + p)^3)$	High
	Interpolating adjoint ◀	Good	$\mathcal{O}(n + p)$	$\mathcal{O}((n + p)^3)$	$\mathcal{O}(nK)$
	Quadrature adjoint	Good	$\mathcal{O}(n + p)$	$\mathcal{O}(n^3 + p)$	High
	Gauss adjoint	Good	$\mathcal{O}(n + p)$	$\mathcal{O}(n^3 + p)$	High
	Gauss adjoint ◀	Good	$\mathcal{O}(n + p)$	$\mathcal{O}(n^3 + p)$	$\mathcal{O}(nK)$

Table 1: Comparison in performance and cost of solver-based methods. Methods that can be checkpointed are indicated with the symbol ◀, with K the total number of checkpoints. The nomenclature of the different adjoint methods here follows the naming in the documentation of `SciMLSensitivity.jl` (Rackauckas et al. 2020).

```

    return ∂L∂θ
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. While the previous example illustrates a manual implementation of the adjoint method, the discrete adjoint method can be directly implemented using reverse AD (Section 3.9.2). In the Julia SciML ecosystem, `ReverseDiffAdjoint` performs reverse AD on the numerical solver via `ReverseDiff.jl`, and `TrackerAdjoint` via `Tracker.jl`. As in the case of reverse AD, checkpointing can be used here.

4.2.2.2 Continuous adjoint method

The continuous adjoint methods 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 reverse AD, 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 (58) involves the terms $f(u, \theta, t)$ and $\frac{\partial h}{\partial u}$, which are both functions of $u(t)$. However, in contrast to the discrete adjoint methods, 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 computation of $\lambda(t)$:

- **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 that allows to define a continuous interpolation, a method known as

continuous Runge-Kutta (Alexe et al. 2009; Wanner et al. 1996).

- **Backsolve.** Solve again the original ODE together with the adjoint as the solution of the following reversed augmented system (Chen et al. 2018):

$$\frac{d}{dt} \begin{bmatrix} u \\ \lambda \\ \frac{dL}{d\theta} \end{bmatrix} = \begin{bmatrix} -f \\ -\frac{\partial f}{\partial u}^T \lambda - \frac{\partial h}{\partial u}^T \\ -\lambda^T \frac{\partial f}{\partial \theta} - \frac{\partial h}{\partial \theta} \end{bmatrix} \quad \begin{bmatrix} u \\ \lambda \\ \frac{dL}{d\theta} \end{bmatrix} (t_1) = \begin{bmatrix} u(t_1) \\ \frac{\partial L}{\partial u(t_1)} \\ \lambda(t_0)^T s(t_0) \end{bmatrix}. \quad (75)$$

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). Implicit methods may be used to ensure stability when solving this system of equations. 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. In practice, this method is hardly stable for most complex (even non-stiff) differential equations.

The following example shows how to implement the continuous adjoint method of the solution of Equation (69) using the backsolve strategy ♣₈.

```
using RecursiveArrayTools

# Augmented dynamics
function f_aug(z, p, t)
    u, λ, L = z
    du = f(u, p, t)
    dλ = ∂f∂u(u, p, t)' * λ
    dL = λ' * ∂f∂p(u, p, t)
    VectorOfArray([du, vec(dλ), vec(dL)])
end

# Solution of original ODE
prob = ODEProblem(f, u0, tspan, p)
sol = solve(prob, Euler(), dt=0.001)

# Final state
u1 = sol.u[end]
z1 = VectorOfArray([u1, [1.0, 0.0], zeros(length(p))])

aug_prob = ODEProblem(f_aug, z1, reverse(tspan), p)
u0_, λ0, dLdp_cont = solve(aug_prob, Euler(), dt=-0.001).u[end]
```

Notice that here we used the final state of the solution u of the ODE as starting point, which is then recalculated in backwards direction (implemented via a negative stepsize $dt=-0.001$).

When dealing with stiff DEs, 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 $u(t)$, then solve for λ in reverse 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 size n of the ODE comes from the fact that we still need to solve the original stiff ODE 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 $\mathcal{O}(n^3 + p)$.

4.2.2.3 Continuous checkpointing

Both interpolating and backsolve adjoint methods can be implemented along with a checkpointing scheme. This can be done by choosing saved points in the forward pass. For the interpolating methods, the interpolation is reconstructed in the backwards pass between two save points. This reduces the total memory requirement of the interpolating method to simply the maximum cost of holding an interpolation between two save points, but requires a total additional computational effort equal to one additional forward pass. In the backsolve variation, the value u in the reverse pass can be corrected to be the saved point, thus resetting the numerical error introduced during the backwards evaluation and thus improving the accuracy.

4.2.2.4 Solving the quadrature

Another computational consideration is how the integral in Equation (60) is numerically evaluated. While one can solve the integral simultaneously with the other equations using an ODE solver, this is only recommended with explicit methods as with implicit methods these additional ODE is of size p and thus can increase the complexity of an implicit solve by $O(p^3)$. The interpolating adjoint and backsolve adjoint method use this ODE solver approach for computing the integrand. On the other side, the quadrature adjoint approach avoids this computational cost by computing the dense solution $\lambda(t)$ and then computing the quadrature

$$\int_{t_0}^{t_1} \left(\frac{\partial h}{\partial \theta} + \lambda^T \frac{\partial f}{\partial \theta} \right) dt \approx \sum_{j=1}^N \omega_j \left(\frac{\partial h}{\partial \theta} + \lambda^T \frac{\partial f}{\partial \theta} \right) (\tau_i) \quad (76)$$

where ω_i, τ_i are the weights and knots of a Gauss-Kronrod quadrature method for numerical integration from `QuadGK.jl` (Gonnet 2012; Laurie 1997). This method results in global error control on the integration and removes the cubic scaling within implicit solvers. Nonetheless, it requires a larger memory cost by storing the adjoint pass continuous solution.

Solvers designed for large implicit systems allow for solving explicit integrals based on the ODE solution simultaneously without including the equations in the ODE evaluation in order to avoid this expense. The Sundials CVODE solver introduced this technique specifically for BDF methods (Hindmarsh et al. 2005). In the Julia `DifferentialEquations.jl` solvers, this can be done using a callback (specifically the numerical integration callbacks from the `DiffEqCallbacks.jl` library). The Gauss adjoint method uses the callback approach to allow for a simultaneous explicit evaluation the integral using Gaussian quadrature, similar to (Norcliffe et al. 2023) but using a different approximation to improve convergence.

These differences in the strategies for computing $u(t)$ and the final quadrature give rise to the set of methods in Table 1. Excluding the forward sensitivity equation which was added for reference, all of these are *the adjoint method* with differences being in the way steps of the adjoint method are approximated, and notably Table 1 shows a general trade-off in stability, performance, and memory across the methods. While the Gauss adjoint achieves good properties according to this chart, the quadrature adjoint notably uses a global error control of the quadrature as opposed to the local error control of the Gauss adjoint, and thus can achieve more robust bounding of the error with respect to user chosen tolerances.

5 Generalizations

In this section, we briefly discuss how the ideas covered in Sections 3 and 4 for first-order ODEs generalize to more complicated systems of DEs.

Notice that the application of all the direct methods (finite differences, AD, complex step differentiation, symbolic differentiation) applies to more general systems of DEs. The fundamental behaviour and implementation of these DP methods does not change, although new considerations about numerical accuracy may need to be taken into account, especially for discrete methods based on unmodified solution processes. The mathematical derivation of continuous methods (forward sensitivity equations and continuous adjoint) covered in Sections 3.6 and 3.8 still applies, although more specific details of the DEs may apply (e.g., inclusion for boundary conditions or other constraints). Regarding discrete adjoint methods, the mathematical formulation covered in Section 3.7 and its connection with reverse AD (Section 3.9.2) applies to more general solvers for DEs.

In the next section we are going to consider the cases of higher-order ODEs, PDEs, and the particular case of chaotic systems of ODEs. Further generalizations of sensitivity methods to other families of DEs include differential-algebraic equations (DAE) (Cao et al. 2002) and stochastic differential equations (SDE) (Li et al. 2020b).

5.1 Higher-order ODEs

Higher-order ODEs are characterized by the presence of second and higher-order time derivatives in the differential equation. A simple example popular in structural design consists in the linear dynamic equations used to model elastic structures given by

$$M \frac{d^2 u}{dt^2} + C \frac{du}{dt} + Ku = F(t, \theta), \quad (77)$$

subject to the initial condition $u(t_0) = u_0 \in \mathbb{R}^n$, $\frac{du}{dt}(t_0) = v_0 \in \mathbb{R}^n$, where $M, C, K \in \mathbb{R}^{n \times n}$ are the mass, damping, and stiffness matrices function of some design parameter θ , respectively, and $F(t, \theta)$ is an external forcing (Jensen et al. 2014; Min et al. 1999).

Just as we did in Section 4.1.1, higher-order ODEs can be transformed to first-order ODEs, after which the same sensitivity methods we discussed in this review can be used. However, there may be reasons why we would prefer to avoid this, such as the existence of more efficient higher-order ODEs solvers, including Nyström methods for the case when $C = 0$ (Butcher et al. 1996; Hairer et al. 2008). In this case, the forward sensitivity equations can be derived using the same strategy explained in Section 3.6:

$$\frac{d}{d\theta} \left(M \frac{d^2 u}{dt^2} + Ku - F(t, \theta) \right) = 0, \quad (78)$$

which results in the forward sensitivity equation for the sensitivity $s(t)$:

$$M \frac{d^2 s}{dt^2} + Ks = \frac{\partial F}{\partial \theta} - \frac{dM}{d\theta} \frac{d^2 u}{dt^2} - \frac{dK}{d\theta} u. \quad (79)$$

Similarly, the same strategy introduced Section 3.8 can be followed to derive the continuous adjoint equation (Kang et al. 2006).

5.2 Partial differential equations

Systems of partial differential equations (PDEs) include derivatives with respect to more than one independent variable. As we discussed in Section 2, PDEs play a central role in mathematics,

physics, and engineering, where these variables are usually associated to time and space. Due to the spatial characteristics of such systems, generally boundary conditions need to be provided besides an initial condition for the solutions. While in Section 3.1.1 we briefly introduced the fundamentals for numerical solvers of ODEs, there is a broader family of numerical methods to solve PDEs. These include the finite element method and the finite volume method, among others (Tadmor 2012). For these methods, a required ingredient is the spatial mesh used to discretize the spatial dimension (Thompson et al. 1998). In the case of the discrete adjoint method, all these methods will result in a series of discrete equations where the adjoint method introduced in Section 3.7 will still apply. Continuous methods require a more careful manipulation of the PDE in order to derive correct sensitivity and adjoint equations.

The method of lines can be used to solve PDEs by applying a semi-discretization in the spacial coordinate and then numerically solve a new system of ODEs (Ascher 2008). This implies that all sensitivity methods for ODEs also apply to PDEs. Let us consider the case of the one-dimensional heat equation

$$\begin{aligned}\frac{\partial u}{\partial t} &= D(x, t) \frac{\partial^2 u}{\partial x^2} & x \in [0, 1], t \in [t_0, t_1] \\ u(x, t_0) &= v(x) \\ u(0, t) &= \alpha(t) \\ u(1, t) &= \beta(t)\end{aligned}\tag{80}$$

with $D(x, t) > 0$ a global diffusivity coefficient. In order to numerically solve this equation, we can define a uniform spatial mesh with coordinates $m\Delta x$, $m = 0, 1, 2, \dots, N$ and $\Delta x = 1/N$. If we call $u_m(t) = u(m\Delta x, t)$ and $D_m(t) = D(m\Delta x, t)$ the values of the solution and the diffusivity evaluated in the fixed points in the mesh, respectively, then we can replace the second order partial derivative in Equation (80) by the corresponding second order finite difference

$$\frac{du_m}{dt} = D_m(t) \frac{u_{m-1} - 2u_m + u_{m+1}}{\Delta x^2}\tag{81}$$

for $m = 1, 2, \dots, N - 1$ (in the boundary we simply have $u_0(t) = \alpha(t)$ and $u_N(t) = \beta(t)$). Now, following this semi-discretization, equation (81) is a system of first-order ODEs of size $N - 1$. Semi-discretized PDEs typically involve large systems of coupled and possibly stiff ODEs subject to some suitable boundary conditions. Explicit calculation of the Jacobian quickly becomes cumbersome and eventually intractable as the spatial dimension and the complexity of the PDE increase. Further improvements can be made by exploiting the fact that the coupling in the ODE is sparse, that is, the temporal derivative depends on the state value of the solution in the neighbouring points in the mesh (see Section 4.1.2.5). PDEs are often also subject to additional time stepping constraints, such as the Courant-Fredrichs-Lewy (CFL) condition, which may limit the maximum time step size and thus increase the number of time steps required to obtain a valid solution (Courant et al. 1967).

Besides the methods of lines which already involves a first discretization of the original PDE, the same recipe introduced in this review to derive the forward sensitivity equations and the continuous adjoint method for ODEs can be employed for PDEs (Giles et al. 2000). Assuming that the diffusivity $D = D(x, t; \theta)$ depends on some design parameter θ , the forward sensitivity equation is obtained by differentiating Equation (80) with respect to θ . This defines a new PDE

$$\frac{\partial s}{\partial t} = D \frac{\partial^2 s}{\partial x^2} + \frac{\partial D}{\partial \theta} \frac{\partial^2 u}{\partial x^2}\tag{82}$$

for the sensitivity $s(x, t) = \frac{d}{d\theta} u(x, t; \theta)$. The continuous adjoint equation can be derived using the strategy followed in Section 3.8 by multiplying the forward sensitivity equation by the transpose of

the adjoint $\lambda(x, t)$ so that we can efficiently compute gradients of the objective function

$$L(\theta) = \int_{t_0}^{t_1} \int_0^1 h(u(x, t; \theta); \theta) dx dt. \quad (83)$$

For the one dimensional heat equation in Equation (80), it is easy to derive using integration by parts the adjoint PDE given by

$$\frac{\partial \lambda}{\partial t} = -D \frac{\partial^2 \lambda}{\partial x^2} - \frac{\partial h^T}{\partial u} \quad (84)$$

with zero final condition $\lambda(x, t_1) \equiv 0$ and boundary conditions $\lambda(0, t) \equiv \lambda(1, t) \equiv 0$ (Duchateau 1996).

An important consideration when working with PDEs is that meshing may be sensitive to model parameters which can lead to errors in the calculation of derivatives (Nadarajah et al. 2000). For example, this is a problem in finite differences since differences in the values of the objective function evaluated at θ and $\theta + \delta\theta$ can be affected by the choice of different meshes. The same errors induced by the adaptive stepsize controller in the case of AD (Section 4.1.2.4) can appear in cases of meshes that do not account for the joint error of the original PDE and its sensitivity. This can produce inaccurate gradients in the case of coarser meshes where the mesh or the numerical solver have an impact on the accuracy of the solution of the PDE (Economou et al. 2017; Kenway et al. 2019).

Independently of how DP is implemented, PDEs remain some of the most challenging problems for computing sensitivities due to the frequent combination of a large number of discretized possibly stiff ODEs, with a large memory footprint. This makes it difficult to strike a balance between memory usage and computational performance. There are, however, numerous recent developments that have made solutions to these challenges more accessible. As it will also be discussed in Section 6, sensitivity methods that require the storage of a dense forward solution need special treatment, such as reverse AD and adjoint methods. Their huge memory footprint can be mitigated by using checkpointing (see Sections 4.1.2.3 and 4.2.2.3). However, the memory requirements for even moderate size PDEs (e.g. 10^2 to 10^3 equations) over long time spans can still incur a large memory cost in cases where many checkpoints are required for stability in the reverse pass. This again can be mitigated by a multi-level checkpointing approach that enables checkpointing to either memory or to disk. Another practical consideration when differentiating numerical PDE solvers arises from the way they are typically implemented. Due to the large size of the system, numerical calculations for PDEs are typically performed in-place, i.e. large memory buffers are often used to store intermediate calculations and system state thereby avoiding the need to repeatedly allocate large amounts of memory for each array operation. This can preclude the use of reverse AD implementations that do not support in-place mutation of arrays. Automated sparsity detection (Gowda et al. 2019) and Newton-Krylov methods (Knoll et al. 2004; Montoison et al. 2023) can drastically decrease both the time and space complexity of calculating JVPs or VJP for large systems. Recent advances in applying AD to implicit functions, i.e. functions which require the solution of a nonlinear system, also provide a promising path forward for many complex PDE problems that often involve multiple nested numerical solvers (Blondel et al. 2022). Finally, some state-of-the-art AD tools such as Enzyme (Moses et al. 2020) are able to support both in-place modification of arrays as well as complex control flow, making them directly applicable to many high efficiency numerical codes for solving PDEs.

5.3 Chaotic systems

Continuous (nonlinear or infinite-dimensional) dynamical systems described by ODEs or PDEs can exhibit chaotic behavior (Strogatz 2018). In contrast to other systems we discussed previously,

chaotic systems appear to become random after a certain, system-specific time scale, called the *Lyapunov time*, making precise future predictions infeasible even though the underlying dynamical description might be completely deterministic. In particular, such systems are characterized by their strong sensitivity to small perturbations of the parameters or initial conditions, i.e. small changes in the initial state or parameter can result in large differences in a later state, which is popularly known under the term *butterfly effect* (Lorenz 1963). As a consequence, all the sensitivity methods discussed in the previous sections become less useful when applied to chaotic systems and special considerations need to be taken under account (Wang et al. 2014a).

The butterfly effect makes inverse modelling based on point evaluations of the trajectory unpractical. Therefore, we here resort to the loss function consisting in the long-time-averaged quantity

$$\langle L(\theta) \rangle_T = \frac{1}{T} \int_0^T h(u(t; \theta), \theta) dt, \quad (85)$$

where $h(u(t; \theta), \theta)$ is the instantaneous loss and $u(t; \theta)$ denotes the state of the dynamical system at time t . In the presence of positive Lyapunov exponents, errors in solutions of the forward sensitivity equations and adjoint method to compute the gradient of $\langle L(\theta) \rangle_T$ with respect to θ blow up (exponentially fast) instead of converging to the actual gradient. To address these issues, various modifications and methods have been proposed, including approaches based on ensemble averages (Eyink et al. 2004; Lea et al. 2000), the Fokker-Planck equation (Blonigan et al. 2014; Thuburn 2005), the fluctuation-dissipation theorem (Abramov et al. 2007, 2008; Leith 1975), shadowing lemma (Blonigan 2017; Blonigan et al. 2018; Ni et al. 2019a, 2017, 2019b; Wang 2013, 2014; Wang et al. 2014b), and modifications of Ruelle’s formula (Chandramoorthy et al. 2022; Ni 2020), which provides closed-form expressions and differentiability conditions for $\langle L(\theta) \rangle_T$ under the assumption of uniform hyperbolic systems (Ruelle 1997, 2009).

In Julia, the following methods based on the shadowing lemma are currently supported in the packages `AdjointLSS`, `ForwardLSS`, `NILSAS`, and `NILSS`. Standard derivative approximations are inappropriate for chaotic systems and will not give convergent estimates when the simulation time is a multiple of the Lyapunov time.

6 Recommendations

There is no sensitivity method that is universally suitable for all types of DE problems and that performs better under all conditions. However, in light of the methods we explore in this work, we can give general guidelines on which methods to use in specific circumstances. In this section we provide a practical guidance of which methods are the most suitable for different situations. A simplified overview of this decision-making process is depicted in Figure 7.

Size of the system

Working with small systems

For sufficiently small systems of less than 50 parameters and ODEs, that is $n + p < 50$, it has been shown that forward AD and forward sensitivity equations are the most efficient methods, outperforming adjoint methods. The original benchmark of these methods is included in (Ma et al. 2021), though the `SciMLBenchmarks` system continually updates the benchmarks and has revised the cutoff point as reverse AD engines improved. See <https://docs.sciml.ai/SciMLBenchmarksOutput/stable/> for continued updates. Furthermore, as we have shown in Section 4.1, AD outperforms other forms of direct differentiation (finite differences, complex-step

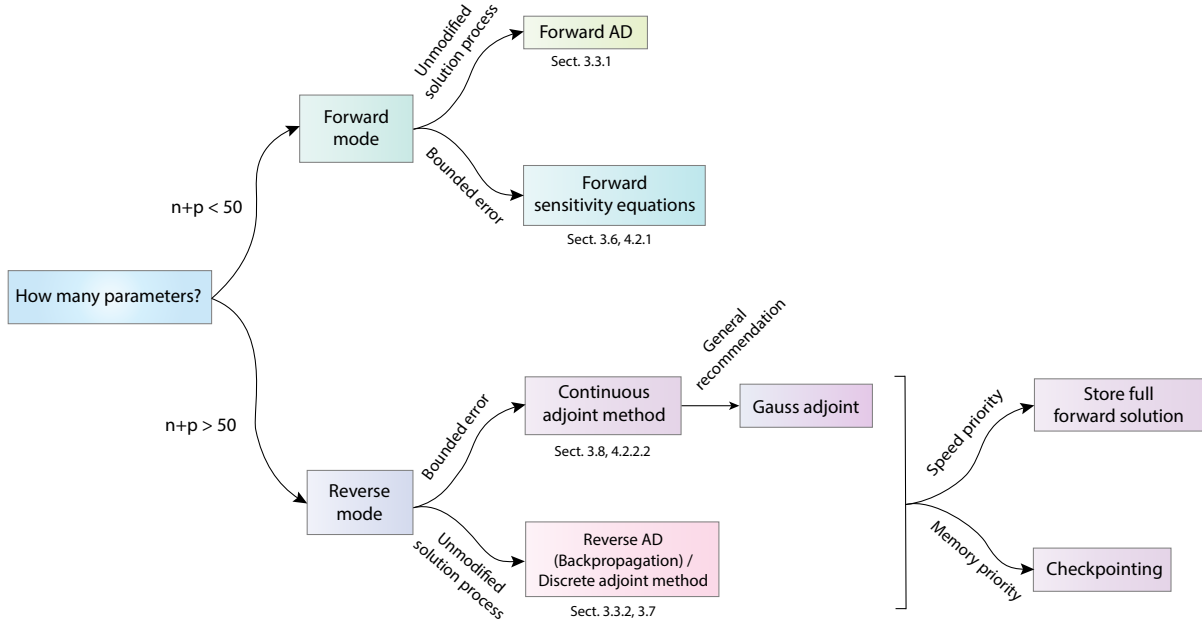


Figure 7: Decision-making tree summarizing the choice of sensitivity methods for different problems depending on: the number of parameters p , the number of ODEs n , the need for an unmodified solution during differentiation vs a bounded error (e.g. in the presence of a numerical solver to ensure correct gradients) and memory-speed trade-off.

differentiation). Modern scientific software commonly support AD, making forward AD the best choice for small problems.

Working with large systems

For larger systems with more than 50 parameters plus size of the ODE, reverse techniques are required. As explained in section 4.2, the continuous adjoint method, particularly the Gauss adjoint, and for very specific cases, the interpolating adjoint and quadrature adjoint, are the most suitable methods to tackle large stiff systems. The choice between these three types of adjoints will be problem-specific and will depend on the trade-off between numerical stability, performance and memory usage. Adjoint methods supporting checkpointing present more flexibility in this sense, and can allow modulating the method depending on the performance vs memory or input/output constraints of each problem.

Unlike for small systems of ODEs and a reduced number of parameters, differentiating large ODEs (e.g. stiff discretized PDEs) with respect to a large number of parameters (e.g., in a neural network or in large-scale inversion), is a much more complex problem. Current state-of-the-art tools can easily work for a wide array of small systems (Rackauckas et al. 2020), whereas methods for large systems are still under heavy development or require tailored approaches and are likely to see many changes and improvements in the future. We refer to Table 1 for considerations of which adjoint method to use depending the stability, performance, and memory trade-offs.

Special considerations for neural networks in DEs. While the general guidelines for large systems hold for computing sensitivities for problems involving neural networks, it might be beneficial to use discrete methods when the training cost for the former is prohibitively high. For example, (Onken

et al. 2020) demonstrates that discrete methods speed up neural ODE training by 6x. Additionally, (Pal et al. 2021) show that discrete methods enable using information from the DE solver to speed up training and inference of neural ODEs and SDEs by 1.4x and 1.8x, respectively. These methods are available in `SciMLSensitivity.jl` via `TrackerAdjoint` and `DiffEqCallbacks.jl`.

Efficiency vs stability

When using discrete methods, it is important to be aware that the differentiation machinery is applied after the numerical solver for the differential equation has been specified, meaning that the derivatives are computed with respect to the time discretization instead of the solution (Eberhard et al. 1996). As discussed in Section 4.1.2.1, this can mean the method is non-convergent in the case where the iterative solver has adaptive stepsize controllers that depend on the parameter to differentiate. Although some solutions have been proposed and implemented in Julia to solve this in the case of discrete methods (Eberhard et al. 1996), this is a problem that continuous methods do not have since they apply the differentiation step before the numerical algorithm has been specified. Using many of the aforementioned tricks, such as continuous checkpointing and Gaussian quadrature approximations, continuous sensitivity analysis tends to be more memory and computationally efficient. However, the errors of discrete adjoint method’s derivative error may better represent the actual code being evaluated. For this reason, discrete adjoint method have been found in some instances to lead to more stable optimizations.

In a nutshell, continuous adjoint methods tend to be more efficient while discrete adjoint methods tend to be more stable (Rackauckas et al. 2020), though the opposite can apply and as such the choice ultimately depends on the nuances of each problem. This is reflected in the fact that discrete methods usually differentiate the unmodified solution of the original ODE, while continuous methods adapt the solution of the original ODE and the sensitivity/adjoint to control for their joint numerical error.

Choosing a direct method

When computing the gradient of a generic function other than a numerical solver, we further recommend the use of AD (reverse or forward depending the number of parameters) as the direct method of choice, outperforming finite differences, complex step differentiation, and symbolic differentiation. This recommendation applies also for the inner JVPs and VJP calculations performed inside the numerical solver (Section 4.2.1.1). As discussed in Section 4.1, finite differences and complex step differentiation do not really provide an advantage over AD in terms of precision and require the tuning of the stepsize ε . On the other hand, if symbolic differentiation can be more efficient in nested cases or when the sparsity pattern of the Jacobian is known, in general this advantage is not drastic in most real cases and can generate difficulties when used inside the numerical solver.

However, this recommendation is constrained by the availability and interoperability of different AD and sensitivity software. For example, when computing higher-order derivatives multiple layers of direct methods became more difficult to implement and may result in complicated computer programs. In this cases, complex step differentiation may offer an interesting alternative with similar performance than AD for small stepsizes. It is important to mention that incorrect implementations of both forward and reverse AD can lead to *perturbation confusion*, an existing problem in some AD software where either repeated applications of AD or differentiation with respect to different dual variables result indistinguishable (Manzyuk et al. 2019; Siskind et al. 2005).

Taking into account model architecture

Code structure and characteristics have a very strong impact in the choice of which packages to use to compute the sensitivities. Within the Julia and Python ecosystems, each available AD package implements a specific AD technique that will face certain limitations. Current limitations include:

- The use of control flow (i.e. `if/else` statements; `for` and `while` loops) presents issues for dynamic (tape-based) AD methods (see Section 4.1.2.2). This is currently not supported by `ReverseDiff.jl` (with tape compilation) and partially supported by JAX in Python. Non-tape-based AD methods tend to support this, like `Enzyme.jl` and `Zygote.jl`.
- Mutation of arrays (i.e. in-place operations) is sometimes problematic, since it does not allow to preserve the chain rule during reverse differentiation. As such, mutations are not possible for packages like `Zygote.jl` or JAX. It is however currently supported by `ReverseDiff.jl` and `Enzyme.jl`.
- Compatibility with GPUs (Graphical Processing Units) is still greatly under development for sensitivity methods. Certain AD packages like `ReverseDiff.jl` do not support GPU operations, while others like JAX, `Enzyme.jl` and `Zygote.jl` support it. This makes the former unsuitable for problems involving large neural networks (e.g, neural ODEs (Chen et al. 2018)) that rely on GPUs for scalability.

It is important to bear in mind that direct methods are easier to implement in programming languages where AD already exists and sometimes does not require any special package, like for the Julia programming language. Nonetheless, users must be aware of the aforementioned convergence issues of AD naively applied to solvers. Thus, we recommend the use of robust and tested software when available (e.g., the Julia SciML ecosystem or `Difffrax` in Python), as the solvers must apply corrections to AD implementations in order to guarantee numerically correct derivatives.

7 Conclusions

We presented a comprehensive overview of the different existing methods for calculating the sensitivity and gradients of functions, including loss functions, 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 differential programming tools 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 vs reverse and discrete vs continuous. We further compare the mathematical and computational foundations of these methods, which we believe enlightens the discussion on sensitivity methods and helps to demystify misconceptions around the sometimes apparent differences between methods. Then, we have shown how these methods can be translated to software implementations, evaluating considerations that we must take into account when implementing or using a sensitivity algorithm. We further exemplified how these methods are implemented in the Julia programming language.

There exists a myriad of options and combinations to compute sensitivities of functions involving differential equations, further complicated by jargon and scientific cultures of different communities. We hope this review paper provides a clearer overview on this topic, and can serve as an entry point to navigate this field and guide researchers in choosing the most appropriate method for their scientific application.

Differentiable programming is opening new ways of doing research across different domains of science and engineering. Arguably, its potential has so far been under-explored but is being rediscovered in the age of data-driven science. Realizing its full potential, requires 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 emerge.

Software availability

All the scripts and code shown in this paper can be found in the GitHub repository <https://github.com/ODINN-SciML/DiffEqSensitivity-Review>. Examples of available code are indicated in the manuscript with the symbol ♣. See Appendix A for a complete description of the scripts provided.

Contribution statement

The following categories are based on the [Contributor Roles Taxonomy \(CRediT\)](#). FSap: conceptualization, investigation, project administration, software, visualization, writing - original draft. JB: conceptualization, visualization, writing - review and editing. FSch: conceptualization, investigation, software, writing - review and editing. BG: conceptualization, software, writing - review and editing. AP: software, writing - review and editing. VB: conceptualization, writing - review and editing. PH: conceptualization, investigation, supervision, writing - review and editing. GH: conceptualization, supervision, writing - review and editing. FP: conceptualization, funding acquisition, supervision, writing - review and editing. PP: conceptualization, supervision, writing - review and editing. CR: conceptualization, funding acquisition, investigation, software, supervision, writing - review and editing.

Acknowledgments

FSap would like to thank Jonathan Taylor, Ryan Giordano, Alexander Strang, and Olivier Bonte for useful comments and feedback. FSap and FP benefits from the Jupyter meets the Earth project supported by the NSF Earth Cube Program under awards 1928406, 1928374. JB has been supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek, Stichting voor de Technische Wetenschappen (Vidi grant 016.Vidi.171.063). BG acknowledges the support of the Helmholtz Einstein International Berlin Research School in Data Science as well as the Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig. PH was supported in part by NSF CSSI #OAC-2103942, ONR #N00014-20-1-2772, DOE #DE-SC002317, and a JPL/Caltech subcontract of the NASA Estimating the Circulation and Climate of the Ocean (ECCO) project. GH was supported by the NSF grant DEB-1933497.

Fsch, AP, and CR works was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and in part by the National Science Foundation under Grant DMS-2309596. This material is based upon work supported by the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0003965. This report was prepared as an account

of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. This material is based upon work supported by the National Science Foundation under grant no. OAC-2103804 , no. OSI-2029670, no. DMS-2325184, no. PHY-2028125. This material was supported by The Research Council of Norway and Equinor ASA through the Research Council project "308817 - Digital wells for optimal production and drainage". Fsch, AP, and CR research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. Fsch, AP, and CR would like to thank DARPA for funding this work through the Automating Scientific Knowledge Extraction and Modeling (ASKEM) program, Agreement No. HR0011262087. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Appendices

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](https://github.com/ODINN-SciML/DiffEqSensitivity-Review).

- ♣₁ **Comparison of direct methods.** The script <https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/DirectMethods/Comparison/direct-comparison.jl> reproduces Figure 6.
- ♣₂ **Dual numbers definition.** The script https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/DirectMethods/DualNumbers/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 using multiple dispatch.
- ♣₃ **When AD is algorithmically correct but numerically wrong.** The script <https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SensitivityForwardAD/example-AD-tolerances.jl> includes the example shown in Section 4.1.2.1 and further elaborated in Section 4.1.2.4 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 and the Julia https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SensitivityForwardAD/testgradient_julia.jl.
- ♣₄ **When AD is algorithmically correct but numerically wrong (JAX).** Python script https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SensitivityForwardAD/testgradient_python.py.
- ♣₅ **Complex step in numerical solver.** The script https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/DirectMethods/ComplexStep/complex_solver.jl shows how to define the dynamics of the ODE to support complex variables and then compute the complex step derivative.
- ♣₆ **Forward sensitivity equation.** The scrip https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SolverMethods/Harmonic/forward_sensitivity_equations.jl includes a manual implementation of the forward sensitivity equations. This also includes how to compute the same sensitivity using ForwardSensitivity in Julia.
- ♣₇ **Discrete adjoint method.** The script https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SolverMethods/Harmonic/adjoint_discrete.jl includes a manual implementation of the discrete adjoint method for the simple harmonic oscillator.
- ♣₈ **Continuous adjoint method.** The script https://github.com/ODINN-SciML/DiffEqSensitivity-Review/blob/main/code/SolverMethods/Harmonic/adjoint_continuous.jl includes a manual implementation of the continuous adjoint method for the simple harmonic oscillator.

References

- 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.
- Abdelhafez, M., B. Baker, A. Gyenis, P. Mundada, A. A. Houck, D. Schuster, and J. Koch (2020). “Universal gates for protected superconducting qubits using optimal control”. In: *Physical Review A* 101.2, p. 022321. DOI: [10.1103/PhysRevA.101.022321](https://doi.org/10.1103/PhysRevA.101.022321).
- Abdelhafez, M., D. I. Schuster, and J. Koch (2019). “Gradient-based optimal control of open quantum systems using quantum trajectories and automatic differentiation”. In: *Phys. Rev. A* 99.5, p. 052327. DOI: [10.1103/PhysRevA.99.052327](https://doi.org/10.1103/PhysRevA.99.052327).
- Abramov, R. V. and A. J. Majda (2007). “Blended response algorithms for linear fluctuation-dissipation for complex nonlinear dynamical systems”. In: *Nonlinearity* 20.12, p. 2793. DOI: <https://iopscience.iop.org/article/10.1088/0951-7715/20/12/004>.
- (2008). “New approximations and tests of linear fluctuation-response for chaotic nonlinear forced-dissipative dynamical systems”. In: *Journal of Nonlinear Science* 18, pp. 303–341. DOI: <https://doi.org/10.1007/s00332-007-9011-9>.
- Aide, T. M., C. Corrada-Bravo, M. Campos-Cerqueira, C. Milan, G. Vega, and R. Alvarez (July 16, 2013). “Real-Time Bioacoustics Monitoring and Automated Species Identification”. In: *PeerJ* 1.1, e103. DOI: [10.7717/peerj.103](https://doi.org/10.7717/peerj.103).
- Åkesson, A., A. Curtsdotter, A. Eklöf, B. Ebenman, J. Norberg, and G. Barabás (Dec. 6, 2021). “The Importance of Species Interactions in Eco-Evolutionary Community Dynamics under Climate Change”. In: *Nature Communications* 12.1, p. 4759. DOI: [10.1038/s41467-021-24977-x](https://doi.org/10.1038/s41467-021-24977-x).
- 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](https://doi.org/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.
- Alsos, I. G., V. Boussange, D. P. Rijal, M. Beaulieu, A. G. Brown, U. Herzsuh, J.-C. Svenning, and L. Pellissier (Nov. 7, 2023). *Using Ancient Sedimentary DNA to Forecast Ecosystem Trajectories under Climate Change*. preprint. In Review. DOI: [10.21203/rs.3.rs-3542192/v1](https://doi.org/10.21203/rs.3.rs-3542192/v1).
- Arrazola, J. M., S. Jahangiri, A. Delgado, J. Ceroni, J. Izaac, A. Száva, U. Azad, R. A. Lang, Z. Niu, O. D. Matteo, R. Moyard, J. Soni, M. Schuld, R. A. Vargas-Hernández, T. Tamayo-Mendoza, C. Y.-Y. Lin, A. Aspuru-Guzik, and N. Killoran (2021). “Differentiable quantum computational chemistry with PennyLane”. In: *arXiv*. DOI: [10.48550/arxiv.2111.09967](https://doi.org/10.48550/arxiv.2111.09967).
- 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](https://doi.org/10.1137/9780898719987).
- Barnosky, A. D., E. A. Hadly, J. Bascompte, E. L. Berlow, J. H. Brown, M. Fortelius, W. M. Getz, J. Harte, A. Hastings, P. A. Marquet, N. D. Martinez, A. Mooers, P. Roopnarine, G. Vermeij, J. W. Williams, R. Gillespie, J. Kitzes, C. Marshall, N. Matzke, D. P. Mindell, E. Revilla, and A. B. Smith (2012). “Approaching a State Shift in Earth’s Biosphere”. In: *Nature* 486.7401, pp. 52–58. DOI: [10.1038/nature11018](https://doi.org/10.1038/nature11018).
- Barton, R. R. (1992). “Computing Forward Difference Derivatives In Engineering Optimization”. In: *Engineering Optimization* 20.3, pp. 205–224. DOI: [10.1080/03052159208941281](https://doi.org/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](https://doi.org/10.1137/0711010).
- Bauer, I., H. G. Bock, S. Körkel, and J. P. Schlöder (2000). “Numerical methods for optimum experimental design in DAE systems”. In: *Journal of Computational and Applied mathematics* 120.1-2, pp. 1–25.

- Bauer, P., A. Thorpe, and G. Brunet (2015). “The quiet revolution of numerical weather prediction”. In: *Nature* 525.7567, pp. 47–55. DOI: [10.1038/nature14956](https://doi.org/10.1038/nature14956).
- Baydin, A. G., B. A. Pearlmutter, A. A. Radul, and J. M. Siskind (2017). “Automatic Differentiation in Machine Learning: A Survey”. In: *J. Mach. Learn. Res.* 18.1, pp. 5595–5637.
- Bell, B. M. and J. V. Burke (2008). “Advances in Automatic Differentiation”. In: *Lecture Notes in Computational Science and Engineering*, pp. 67–77. DOI: [10.1007/978-3-540-68942-3_7](https://doi.org/10.1007/978-3-540-68942-3_7).
- 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](https://doi.org/10.1147/rd.176.0525).
- Betancourt, M. (2017). “A Conceptual Introduction to Hamiltonian Monte Carlo”. In: *arXiv*. DOI: [10.48550/arxiv.1701.02434](https://doi.org/10.48550/arxiv.1701.02434).
- 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](https://doi.org/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](https://doi.org/10.48550/arxiv.1209.5145).
- Blessing, S., T. Kaminski, F. Lunkeit, I. Matei, R. Giering, A. Köhl, M. Scholze, P. Herrmann, K. Fraedrich, and D. Stammer (2014). “Testing variational estimation of process parameters and initial conditions of an earth system model”. In: *Tellus A* 66.0, p. 22606. DOI: [10.3402/tellusa.v66.22606](https://doi.org/10.3402/tellusa.v66.22606).
- Blondel, M., Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-Lopez, F. Pedregosa, and J.-P. Vert (2022). “Efficient and Modular Implicit Differentiation”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 5230–5242.
- Blondel, M. and V. Roulet (2024). “The elements of differentiable programming”. In: *arXiv*. DOI: [10.48550/arXiv.2403.14606](https://doi.org/10.48550/arXiv.2403.14606).
- Blonigan, P. J. (2017). “Adjoint sensitivity analysis of chaotic dynamical systems with non-intrusive least squares shadowing”. In: *Journal of Computational Physics* 348, pp. 803–826. DOI: <https://doi.org/10.1016/j.jcp.2017.08.002>.
- Blonigan, P. J. and Q. Wang (2014). “Probability density adjoint for sensitivity analysis of the mean of chaos”. In: *Journal of Computational Physics* 270, pp. 660–686. DOI: <https://doi.org/10.1016/j.jcp.2014.04.027>.
- (2018). “Multiple shooting shadowing for sensitivity analysis of chaotic dynamical systems”. In: *Journal of Computational Physics* 354, pp. 447–475. DOI: <https://doi.org/10.1016/j.jcp.2017.10.032>.
- Bockhorn, A., S. H. K. Narayanan, and A. Walther (2020). “2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing”. In: pp. 22–31. DOI: [10.1137/1.9781611976229.3](https://doi.org/10.1137/1.9781611976229.3).
- Bolibar, J., F. Sapienza, F. Maussion, R. Lguensat, B. Wouters, and F. Pérez (2023). “Universal differential equations for glacier ice flow modelling”. In: *Geoscientific Model Development* 16.22, pp. 6671–6687. DOI: [10.5194/gmd-16-6671-2023](https://doi.org/10.5194/gmd-16-6671-2023).
- Borowiec, M. L., R. B. Dikow, P. B. Frandsen, A. McKeeken, G. Valentini, and A. E. White (Aug. 2022). “Deep Learning as a Tool for Ecology and Evolution”. In: *Methods in Ecology and Evolution* 13.8, pp. 1640–1660. DOI: [10.1111/2041-210X.13901](https://doi.org/10.1111/2041-210X.13901).
- Boussange, V., P. V. Aceituno, F. Schäfer, and L. Pellissier (2024). “Partitioning time series to improve process-based models with machine learning”. In: *bioRxiv*. DOI: [10.1101/2022.07.25.501365](https://doi.org/10.1101/2022.07.25.501365).
- Boussange, V., S. Becker, A. Jentzen, B. Kuckuck, and L. Pellissier (Dec. 1, 2023). “Deep Learning Approximations for Non-Local Nonlinear PDEs with Neumann Boundary Conditions”. In: *Partial Differential Equations and Applications* 4.6, p. 51. DOI: [10.1007/s42985-023-00244-0](https://doi.org/10.1007/s42985-023-00244-0).
- Boussange, V. and L. Pellissier (Dec. 6, 2022). “Eco-Evolutionary Model on Spatial Graphs Reveals How Habitat Structure Affects Phenotypic Differentiation”. In: *Communications Biology* 5.1, p. 668. DOI: [10.1038/s42003-022-03595-3](https://doi.org/10.1038/s42003-022-03595-3).
- Bradley, A. M. (2013). *PDE-constrained optimization and the adjoint method*. Tech. rep. Technical Report. Stanford University.
- Brézis, H. (2011). *Functional analysis, Sobolev spaces and partial differential equations*. Vol. 2. 3. Springer.
- Brown, K. S. and J. P. Sethna (2003). “Statistical mechanical approaches to models with many poorly known parameters”. In: *Physical review E* 68.2, p. 021904.

- 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](https://doi.org/10.1109/TSMC.1979.4310229).
- Buizza, R. and T. N. Palmer (1995). “The singular-vector structure of the atmospheric global circulation”. In: *Journal of the Atmospheric Sciences* 52.9, pp. 1434–1456. DOI: [10.1175/1520-0469\(1995\)052<1434:tvsot>2.0.co;2](https://doi.org/10.1175/1520-0469(1995)052<1434:tvsot>2.0.co;2).
- 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](https://doi.org/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](https://doi.org/10.1016/s0168-9274(96)00048-7).
- Cao, J., G. F. Fussmann, and J. O. Ramsay (2008). “Estimating a predator-prey dynamical model with the parameter cascades method”. In: *Biometrics* 64.3, pp. 959–967.
- 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](https://doi.org/10.1016/s0377-0427(02)00528-9).
- Chalmandrier, L., F. Hartig, D. C. Laughlin, H. Lischke, M. Pichler, D. B. Stouffer, and L. Pellissier (Dec. 2021). “Linking Functional Traits and Demography to Model Species-Rich Communities”. In: *Nature Communications* 12.1, p. 2724. DOI: [10.1038/s41467-021-22630-1](https://doi.org/10.1038/s41467-021-22630-1).
- Chandramoorthy, N. and Q. Wang (2022). “Efficient computation of linear response of chaotic attractors with one-dimensional unstable manifolds”. In: *SIAM Journal on Applied Dynamical Systems* 21.2, pp. 735–781. DOI: <https://doi.org/10.1137/21M1405599>.
- Chen, R. T. Q. (2018). *torchdiffeq*.
- Chen, R. T., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud (2018). “Neural ordinary differential equations”. In: *Advances in neural information processing systems* 31.
- Chen, S., A. Shojai, and D. M. Witten (2017). “Network reconstruction from high-dimensional ordinary differential equations”. In: *Journal of the American Statistical Association* 112.520, pp. 1697–1707.
- Chen, S., Z. Lyu, G. K. Kenway, and J. R. Martins (2016). “Aerodynamic shape optimization of common research model wing-body-tail configuration”. In: *Journal of Aircraft* 53.1, pp. 276–293.
- Christianson, B. (1994). “Reverse accumulation and attractive fixed points”. In: *Optimization Methods and Software* 3.4, pp. 311–326.
- (1998). “Reverse accumulation and implicit functions”. In: *Optimization Methods and Software* 9.4, pp. 307–322.
- Clifford (1871). “Preliminary sketch of biquaternions”. In: *Proceedings of the London Mathematical Society* 1.1, pp. 381–395.
- Colijn, C., A. Fowler, and M. C. Mackey (2006). “High frequency spikes in long period blood cell oscillations”. In: *Journal of mathematical biology* 53, pp. 499–519.
- Courant, R., K. Friedrichs, and H. Lewy (1967). “On the Partial Difference Equations of Mathematical Physics”. In: *IBM journal of Research and Development* 11.2, pp. 215–234.
- Courtier, P. and O. Talagrand (1987). “Variational Assimilation of Meteorological Observations With the Adjoint Vorticity Equation. II: Numerical Results”. In: *Quarterly Journal of the Royal Meteorological Society* 113.478, pp. 1329–1347. DOI: [10.1002/qj.49711347813](https://doi.org/10.1002/qj.49711347813).
- 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](https://doi.org/10.1098/rsta.2016.0153).
- 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](https://doi.org/10.1073/pnas.1912789117).
- Curtsdotter, A., H. T. Banks, J. E. Banks, M. Jonsson, T. Jonsson, A. N. Laubmeier, M. Traugott, and R. Bommarco (Feb. 7, 2019). “Ecosystem Function in Predator–Prey Food Webs—Confronting Dynamic Models with Empirical Data”. In: *Journal of Animal Ecology* 88.2. Ed. by D. Stouffer, pp. 196–210. DOI: [10.1111/1365-2656.12892](https://doi.org/10.1111/1365-2656.12892).
- Dahlquist, G. (1985). “33 years of numerical instability, Part I”. In: *BIT Numerical Mathematics* 25.1, pp. 188–204. DOI: [10.1007/bf01934997](https://doi.org/10.1007/bf01934997).

- Dai, X. and L. Li (2022). “Kernel ordinary differential equations”. In: *Journal of the American Statistical Association* 117.540, pp. 1711–1725.
- Dalle, G. and A. Hill (2024). *DifferentiationInterface.jl*. Version DifferentiationInterfaceTest-v0.4.4. DOI: [10.5281/zenodo.11573435](https://doi.org/10.5281/zenodo.11573435).
- Dandekar, R., C. Rackauckas, and G. Barbastathis (2020). “A Machine Learning-Aided Global Diagnostic and Comparative Tool to Assess Effect of Quarantine Control in COVID-19 Spread”. In: *Patterns* 1.9, p. 100145. DOI: [10.1016/j.patter.2020.100145](https://doi.org/10.1016/j.patter.2020.100145).
- Der Houwen, P. J. van and B. P. Sommeijer (1980). “On the internal stability of explicit, m-stage Runge-Kutta methods for large m-values”. In: *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 60.10, pp. 479–485.
- Dhert, T., T. Ashuri, and J. R. Martins (2017). “Aerodynamic shape optimization of wind turbine blades using a Reynolds-averaged Navier–Stokes model and an adjoint method”. In: *Wind Energy* 20.5, pp. 909–926.
- Dimet, F.-X. L., I. M. Navon, and D. N. Daescu (2002). “Second-Order Information in Data Assimilation*”. In: *Monthly Weather Review* 130.3, pp. 629–648. DOI: [10.1175/1520-0493\(2002\)130<0629:soida>2.0.co;2](https://doi.org/10.1175/1520-0493(2002)130<0629:soida>2.0.co;2).
- Ding, A. (2000). “H. Wu. A comparison study of models and fitting procedures for biphasic viral decay rates in viral dynamic models”. In: *Biometrics* 56, pp. 16–23.
- Dorigo, T., A. Giammanco, P. Vischia, M. Aehle, M. Bawaj, A. Boldyrev, P. d. C. Manzano, D. Derkach, J. Donini, A. Edelen, F. Fanzago, N. R. Gauger, C. Glaser, A. G. Baydin, L. Heinrich, R. Keidel, J. Kieseler, C. Krause, M. Lagrange, M. Lamparth, L. Layer, G. Maier, F. Nardi, H. E. S. Pettersen, A. Ramos, F. Ratnikov, D. Röhrich, R. R. d. Austri, P. M. R. d. Árbol, O. Savchenko, N. Simpson, G. C. Strong, A. Taliercio, M. Tosi, A. Ustyuzhanin, and H. Zaraket (2022). “Toward the End-to-End Optimization of Particle Physics Instruments with Differentiable Programming: a White Paper”. In: *arXiv*. DOI: [10.48550/arxiv.2203.13818](https://doi.org/10.48550/arxiv.2203.13818).
- Dormann, C. F. (Sept. 3, 2007). “Promising the Future? Global Change Projections of Species Distributions”. In: *Basic and Applied Ecology* 8.5, pp. 387–397. DOI: [10.1016/j.baae.2006.11.001](https://doi.org/10.1016/j.baae.2006.11.001).
- Duchateau, P. (1996). “An introduction to inverse problems in partial differential equations for physicists, scientists and engineers”. In: *Water Science and Technology Library* 23, pp. 3–3.
- 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](https://doi.org/10.1023/a:1015523018029).
- Dutkiewicz, S., M. J. Follows, P. Heimbach, and J. Marshall (2006). “Controls on ocean productivity and air-sea carbon flux: An adjoint model sensitivity study”. In: *Geophysical Research Letters* 33.2, pp. 159–4. DOI: [10.1029/2005gl024987](https://doi.org/10.1029/2005gl024987).
- 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](https://doi.org/10.1090/s0025-5718-99-01027-3).
- Economon, T. D., J. J. Alonso, T. A. Albring, and N. R. Gauger (2017). “Adjoint formulation investigations of benchmark aerodynamic design cases in su2”. In: *35th AIAA Applied Aerodynamics Conference*, p. 4363.
- 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](https://doi.org/10.1145/3236765).
- Elliott, J. and J. Peraire (1996). “Aerodynamic design using unstructured meshes”. In: *Fluid Dynamics Conference*. This has an example of the hardcore adjoint method implemented for aerodynamics. It may help to read this to see how the adjoint equations is being solved and the size of the problem. DOI: [10.2514/6.1996-1941](https://doi.org/10.2514/6.1996-1941).
- Errico, R. M. (1997). “What is an adjoint model?” In: *Bulletin of the American Meteorological Society* 78.11, pp. 2577–2592.
- Eyink, G., T. Haine, and D. Lea (2004). “Ruelle’s linear response formula, ensemble adjoint schemes and Lévy flights”. In: *Nonlinearity* 17.5, p. 1867.
- Farrell, B. (1988). “Optimal Excitation of Neutral Rossby Waves”. In: *Journal of the Atmospheric Sciences* 45.2, pp. 163–172. DOI: [10.1175/1520-0469\(1988\)045<0163:oeonrw>2.0.co;2](https://doi.org/10.1175/1520-0469(1988)045<0163:oeonrw>2.0.co;2).

- Farrell, B. F. and P. J. Ioannou (1996). “Generalized Stability Theory. Part I: Autonomous Operators”. In: *Journal of the Atmospheric Sciences* 53.14, pp. 2025–2040. DOI: [10.1175/1520-0469\(1996\)053<2025:gstpia>2.0.co;2](https://doi.org/10.1175/1520-0469(1996)053<2025:gstpia>2.0.co;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](https://doi.org/10.1137/120873558).
- Ferreira, D., J. Marshall, and P. Heimbach (2005). “Estimating Eddy Stresses by Fitting Dynamics to Observations Using a Residual-Mean Ocean Circulation Model and Its Adjoint”. In: *Journal of Physical Oceanography* 35.10, pp. 1891–1910. DOI: [10.1175/jpo2785.1](https://doi.org/10.1175/jpo2785.1).
- Fike, J. A. (2013). “Multi-objective optimization using hyper-dual numbers”. PhD thesis.
- Forget, G., J.-M. Campin, P. Heimbach, C. N. Hill, R. M. Ponte, and C. Wunsch (2015). “ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation”. In: *Geoscientific Model Development* 8.10, pp. 3071–3104. DOI: [10.5194/gmd-8-3071-2015](https://doi.org/10.5194/gmd-8-3071-2015).
- Fornberg, B. (1988). “Generation of Finite Difference Formulas on Arbitrarily Spaced Grids”. In: *Mathematics of Computation* 51.184, pp. 699–706.
- Frank, S. A. (2022). “Automatic Differentiation and the Optimization of Differential Equation Models in Biology”. In: *Frontiers in Ecology and Evolution* 10.
- Franklin, O., S. P. Harrison, R. Dewar, C. E. Farrior, Å. Brännström, U. Dieckmann, S. Pietsch, D. Falster, W. Cramer, M. Loreau, H. Wang, A. Mäkelä, K. T. Rebel, E. Meron, S. J. Schymanski, E. Rovenskaya, B. D. Stocker, S. Zaehle, S. Manzoni, M. van Oijen, I. J. Wright, P. Ciais, P. M. van Bodegom, J. Peñuelas, F. Hofhansl, C. Terrer, N. A. Soudzilovskaia, G. Midgley, and I. C. Prentice (2020). “Organizing Principles for Vegetation Dynamics”. In: *Nature Plants* 6.5, pp. 444–453. DOI: [10.1038/s41477-020-0655-x](https://doi.org/10.1038/s41477-020-0655-x).
- Freund, J. B. (2010). “Adjoint-based optimization for understanding and suppressing jet noise”. In: *Procedia Engineering* 6. IUTAM Symposium on Computational Aero-Acoustics for Aircraft Noise Prediction, pp. 54–63. DOI: <https://doi.org/10.1016/j.proeng.2010.09.007>.
- Frolov, S., C. S. Rousseaux, T. Auligne, D. Dee, R. Gelaro, P. Heimbach, I. Simpson, and L. Slivinski (2023). “Road Map for the Next Decade of Earth System Reanalysis in the United States”. In: *Bulletin of the American Meteorological Society* 104.3, E706–E714. DOI: [10.1175/bams-d-23-0011.1](https://doi.org/10.1175/bams-d-23-0011.1).
- Fussmann, G. F., S. P. Ellner, K. W. Shertzer, and N. G. Hairston Jr (2000). “Crossing the Hopf bifurcation in a live predator-prey system”. In: *Science* 290.5495, pp. 1358–1360.
- Gábor, A. and J. R. Banga (Dec. 29, 2015). “Robust and Efficient Parameter Estimation in Dynamic Models of Biological Systems”. In: *BMC Systems Biology* 9.1, p. 74. DOI: [10.1186/s12918-015-0219-2](https://doi.org/10.1186/s12918-015-0219-2).
- Gaikwad, S. S., L. Hascoet, S. H. K. Narayanan, L. Curry-Logan, R. Greve, and P. Heimbach (2023). “SICOPOLIS-AD v2: tangent linear and adjoint modeling framework for ice sheet modeling enabled by automatic differentiation tool Tapenade”. In: *Journal of Open Source Software* 8.83, p. 4679. DOI: [10.21105/joss.04679](https://doi.org/10.21105/joss.04679).
- Gaikwad, S. S., S. H. K. Narayanan, L. Hascoet, J.-M. Campin, H. Pillar, A. Nguyen, J. Hückelheim, P. Hovland, and P. Heimbach (2024). “MITgcm-AD v2: Open source tangent linear and adjoint modeling framework for the oceans and atmosphere enabled by the Automatic Differentiation tool Tapenade”. In: *arXiv*.
- GBIF: The Global Biodiversity Information Facility (2022). *What Is GBIF?* URL: <https://www.gbif.org/what-is-gbif>.
- Geary, W. L., M. Bode, T. S. Doherty, E. A. Fulton, D. G. Nimmo, A. I. T. Tulloch, V. J. D. Tulloch, and E. G. Ritchie (Nov. 14, 2020). “A Guide to Ecosystem Models and Their Environmental Applications”. In: *Nature Ecology & Evolution* 4.11, pp. 1459–1471. DOI: [10.1038/s41559-020-01298-8](https://doi.org/10.1038/s41559-020-01298-8).
- Gebremedhin, A. H., F. Manne, and A. Pothén (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](https://doi.org/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.

- Georgieva, N. K., S. Glavic, M. H. Bakr, and J. W. Bandler (2002). “Feasible Adjoint Sensitivity Technique for Em Design Optimization”. In: *IEEE Transactions on Microwave Theory and Techniques* 50.12, p. 2751. DOI: [10.1109/tmtt.2002.805131](https://doi.org/10.1109/tmtt.2002.805131).
- 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](https://doi.org/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](https://doi.org/10.1145/293686.293695).
- Giering, R., T. Kaminski, R. Todling, R. Errico, R. Gelaro, and N. Winslow (2006). “Automatic Differentiation: Applications, Theory, and Implementations”. In: *Lecture Notes in Computational Science and Engineering*, pp. 275–284. DOI: [10.1007/3-540-28438-9_24](https://doi.org/10.1007/3-540-28438-9_24).
- Giles, M. B. and N. A. Pierce (2000). “An Introduction to the Adjoint Approach to Design”. In: *Flow, Turbulence and Combustion* 65.3–4, pp. 393–415. DOI: [10.1023/a:1011430410075](https://doi.org/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](https://doi.org/10.1016/j.cma.2021.113810).
- Godwin, C. M., F.-H. Chang, and B. J. Cardinale (2020). “An Empiricist’s Guide to Modern Coexistence Theory for Competitive Communities”. In: *Oikos* 129.8, pp. 1109–1127. DOI: [10.1111/oik.06957](https://doi.org/10.1111/oik.06957).
- Goerz, M. H., S. C. Carrasco, and V. S. Malinovsky (2022). “Quantum optimal control via semi-automatic differentiation”. In: *Quantum* 6, p. 871. DOI: [10.22331/q-2022-12-07-871](https://doi.org/10.22331/q-2022-12-07-871).
- Goldberg, D. N., T. A. Smith, S. H. K. Narayanan, P. Heimbach, and M. Morlighem (2020). “Bathymetric Influences on Antarctic Ice-Shelf Melt Rates”. In: *Journal of Geophysical Research: Oceans* 125.11, p. C232. DOI: [10.1029/2020jc016370](https://doi.org/10.1029/2020jc016370).
- 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](https://doi.org/10.1145/103162.103163).
- Goldberg, D. 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.
- Gonnet, P. (2012). “A review of error estimation in adaptive quadrature”. In: *ACM Computing Surveys (CSUR)* 44.4, pp. 1–36.
- Gowda, S., Y. Ma, A. Cheli, M. Gwóźdz, 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](https://doi.org/10.1145/3511528.3511535).
- Gowda, S., Y. Ma, V. Churavy, A. Edelman, and C. Rackauckas (2019). “Sparsity Programming: Automated Sparsity-Aware Optimizations in Differentiable Programming”. In: *Program Transformations for ML Workshop at NeurIPS 2019*.
- 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 (2000). “Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation”. In: *ACM Transactions on Mathematical Software (TOMS)* 26.1, pp. 19–45. DOI: [10.1145/347837.347846](https://doi.org/10.1145/347837.347846).
- (2008). *Evaluating Derivatives*. DOI: [10.1137/1.9780898717761](https://doi.org/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.
- Gunzburger, M. D. (2002). “Perspectives in Flow Control and Optimization”. In: pp. 101–142. DOI: [10.1137/1.9780898718720.ch4](https://doi.org/10.1137/1.9780898718720.ch4).
- 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](https://doi.org/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.
- Hartig, F., J. Dyke, T. Hickler, S. I. Higgins, R. B. O’Hara, S. Scheiter, and A. Huth (Dec. 2012). “Connecting Dynamic Vegetation Models to Data - an Inverse Perspective: Dynamic Vegetation Models - an Inverse

- Perspective”. In: *Journal of Biogeography* 39.12, pp. 2240–2252. DOI: [10.1111/j.1365-2699.2012.02745.x](https://doi.org/10.1111/j.1365-2699.2012.02745.x).
- Hascoet, L. and V. Pasqual (2013). “The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification”. In: *ACM Transactions on Mathematical Software* 39.3, pp. 1–43. DOI: [10.1145/2450153.2450158](https://doi.org/10.1145/2450153.2450158).
- 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.
- Heimbach, P., D. Menemenlis, M. Losch, J.-M. Campin, and C. Hill (2010). “On the formulation of sea-ice models. Part 2: Lessons from multi-year adjoint sea-ice export sensitivities through the Canadian Arctic Archipelago”. In: *Ocean Modelling* 33.1-2, pp. 145–158. DOI: [10.1016/j.ocemod.2010.02.002](https://doi.org/10.1016/j.ocemod.2010.02.002).
- 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.
- Heimbach, P., C. Hill, and R. Giering (2005). “An efficient exact adjoint of the parallel MIT General Circulation Model, generated via automatic differentiation”. In: *Future Generation Computer Systems* 21.8, pp. 1356–1371. DOI: [10.1016/j.future.2004.11.010](https://doi.org/10.1016/j.future.2004.11.010).
- Heimbach, P. and M. Losch (2012). “Adjoint sensitivities of sub-ice-shelf melt rates to ocean circulation under the Pine Island Ice Shelf, West Antarctica”. In: *Annals of Glaciology* 53.60, pp. 59–69. DOI: [10.3189/2012/aog60a025](https://doi.org/10.3189/2012/aog60a025).
- Hey, T., S. Tansley, K. Tolle, and J. Gray (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.
- Higgins, S. I., S. Scheiter, and M. Sankaran (June 2010). “The Stability of African Savannas: Insights from the Indirect Estimation of the Parameters of a Dynamic Model”. In: *Ecology* 91.6, pp. 1682–1692. DOI: [10.1890/08-1368.1](https://doi.org/10.1890/08-1368.1).
- 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.
- Hodgkin, A. L. and A. F. Huxley (1952). “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4, p. 500.
- Hooker, G. (2009). “Forcing function diagnostics for nonlinear dynamics”. In: *Biometrics* 65.3, pp. 928–936.
- Hooker, G. and S. P. Ellner (2015). “Goodness of fit in nonlinear dynamics: Misspecified rates or misspecified states?” In: *The Annals of Applied Statistics* 9.2, pp. 754–776. DOI: [10.1214/15-AOAS828](https://doi.org/10.1214/15-AOAS828).
- Hu, R. (2010). “Supersonic biplane design via adjoint method”. PhD thesis.
- Innes, M. (2018). “Don’t Unroll Adjoint: Differentiating SSA-Form Programs”. In: *arXiv*.
- 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](https://doi.org/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](https://doi.org/10.1080/00029890.1998.12004985).
- Isaac, T., N. Petra, G. Stadler, and O. Ghattas (2015). “Scalable and efficient algorithms for the propagation of uncertainty from data through inference to prediction for large-scale problems, with application to flow of the Antarctic ice sheet”. In: *Journal of Computational Physics* 296.C, pp. 348–368. DOI: [10.1016/j.jcp.2015.04.047](https://doi.org/10.1016/j.jcp.2015.04.047).
- Jameson, A. (1988). “Aerodynamic design via control theory”. In: *Journal of Scientific Computing* 3.3, pp. 233–260. DOI: [10.1007/bf01061285](https://doi.org/10.1007/bf01061285).
- (2003). “Aerodynamic shape optimization using the adjoint method”. In: *Lectures at the Von Karman Institute, Brussels*.
- 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](https://doi.org/10.1007/s00158-013-1024-4).
- Jetz, W., M. A. McGeoch, R. Guralnick, S. Ferrier, J. Beck, M. J. Costello, M. Fernandez, G. N. Geller, P. Keil, C. Merow, C. Meyer, F. E. Muller-Karger, H. M. Pereira, E. C. Regan, D. S. Schmeller, and

- E. Turak (2019). “Essential Biodiversity Variables for Mapping and Monitoring Species Populations”. In: *Nature Ecology and Evolution* 3.4, pp. 539–551. DOI: [10.1038/s41559-019-0826-1](https://doi.org/10.1038/s41559-019-0826-1).
- Jirari, H. (2009). “Optimal control approach to dynamical suppression of decoherence of a qubit”. In: *Europhysics Letters* 87.4, p. 40003. DOI: [10.1209/0295-5075/87/40003](https://doi.org/10.1209/0295-5075/87/40003).
- (2019). “From quantum optimal control theory to coherent destruction of tunneling”. In: *The European Physical Journal B* 92, pp. 1–8. DOI: [10.1140/epjb/e2018-90231-5](https://doi.org/10.1140/epjb/e2018-90231-5).
- Johnson, J. B. and K. S. Omland (Feb. 2004). “Model Selection in Ecology and Evolution”. In: *Trends in Ecology & Evolution* 19.2, pp. 101–108. DOI: [10.1016/j.tree.2003.10.013](https://doi.org/10.1016/j.tree.2003.10.013).
- 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.
- Jouvet, G., G. Cordonnier, B. Kim, M. Lüthi, A. Vieli, and A. Aschwanden (2021). “Deep learning speeds up ice flow modelling by several orders of magnitude”. In: *Journal of Glaciology*, pp. 1–14. DOI: [10.1017/jog.2021.120](https://doi.org/10.1017/jog.2021.120).
- Juedes, D. W. (1991). *A taxonomy of automatic differentiation tools*. Tech. rep. Argonne National Lab., IL (United States).
- Kang, B.-S., G.-J. Park, and J. S. Arora (2006). “A review of optimization of structures subjected to transient loads”. In: *Structural and Multidisciplinary Optimization* 31, pp. 81–95.
- 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](https://doi.org/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](https://doi.org/10.1038/s42254-021-00314-5).
- Kenway, G. K., C. A. Mader, P. He, and J. R. Martins (2019). “Effective adjoint approaches for computational fluid dynamics”. In: *Progress in Aerospace Sciences* 110, p. 100542. DOI: <https://doi.org/10.1016/j.paerosci.2019.05.002>.
- Keulen, F. v., R. Haftka, and N. Kim (2005). “Review of options for structural design sensitivity analysis. Part 1: Linear systems”. In: *Computer Methods in Applied Mechanics and Engineering* 194.30–33, pp. 3213–3243. DOI: [10.1016/j.cma.2005.02.002](https://doi.org/10.1016/j.cma.2005.02.002).
- Kidger, P. (2021). “On Neural Differential Equations”. PhD thesis. University of Oxford.
- 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](https://doi.org/10.1063/5.0060697).
- Kingma, D. P. and J. Ba (Dec. 22, 2014). “Adam: A Method for Stochastic Optimization”.
- Knoll, D. A. and D. E. Keyes (2004). “Jacobian-Free Newton–Krylov Methods: A Survey of Approaches and Applications”. In: *Journal of Computational Physics* 193.2, pp. 357–397. DOI: [10.1016/j.jcp.2003.08.010](https://doi.org/10.1016/j.jcp.2003.08.010).
- Koch, C. P., U. Boscain, T. Calarco, G. Dirr, S. Filipp, S. J. Glaser, R. Kosloff, S. Montangero, T. Schulte-Herbrüggen, D. Sugny, et al. (2022). “Quantum optimal control in quantum technologies. Strategic report on current status, visions and goals for research in Europe”. In: *EPJ Quantum Technology* 9.1, p. 19.
- Kochkov, D., J. Yuval, I. Langmore, P. Norgaard, J. Smith, G. Mooers, J. Lottes, S. Rasp, P. Düben, M. Klöwer, S. Hatfield, P. Battaglia, A. Sanchez-Gonzalez, M. Willson, M. P. Brenner, and S. Hoyer (2023). “Neural General Circulation Models for Weather and Climate”. In: *arXiv*. DOI: [10.48550/arxiv.2311.07222](https://doi.org/10.48550/arxiv.2311.07222).
- Kröger, J., N. Kühl, and T. Rung (2018). “Adjoint volume-of-fluid approaches for the hydrodynamic optimisation of ships”. In: *Ship Technology Research* 65.1, pp. 47–68. DOI: [10.1080/09377255.2017.1411001](https://doi.org/10.1080/09377255.2017.1411001).
- 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](https://doi.org/10.1371/journal.pcbi.1008462).

- Lai, C.-Y., P. Hassanzadeh, A. Sheshadri, M. Sonnewald, R. Ferrari, and V. Balaji (2024). “Machine learning for climate physics and simulations”. In: *arXiv preprint arXiv:2404.13227*.
- Langland, R. H. and N. L. Baker (2004). “Estimation of observation impact using the NRL atmospheric variational data assimilation adjoint system”. In: *Tellus A: Dynamic Meteorology and Oceanography* 56.3, pp. 189–201. DOI: [10.3402/tellusa.v56i3.14413](https://doi.org/10.3402/tellusa.v56i3.14413).
- 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](https://doi.org/10.1145/2168773.2168774).
- Laue, S. (2019). *On the Equivalence of Forward Mode Automatic Differentiation and Symbolic Differentiation*. DOI: [10.48550/ARXIV.1904.02990](https://doi.org/10.48550/ARXIV.1904.02990).
- Laurie, D. (1997). “Calculation of Gauss-Kronrod quadrature rules”. In: *Mathematics of Computation* 66.219, pp. 1133–1145.
- Lea, D. J., M. R. Allen, and T. W. Haine (2000). “Sensitivity analysis of the climate of a chaotic system”. In: *Tellus A: Dynamic Meteorology and Oceanography* 52.5, pp. 523–532. DOI: <https://doi.org/10.3402/tellusa.v52i5.12283>.
- Lea, D. J., T. W. N. Haine, M. R. Allen, and J. A. Hansen (2002). “Sensitivity analysis of the climate of a chaotic ocean circulation model”. In: *Quarterly Journal of the Royal Meteorological Society* 128.586, pp. 2587–2605. DOI: [10.1256/qj.01.180](https://doi.org/10.1256/qj.01.180).
- LeCun, Y., Y. Bengio, and G. Hinton (May 27, 2015). “Deep Learning”. In: *Nature* 521.7553, pp. 436–444. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539).
- Leis, J. R. and M. A. Kramer (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](https://doi.org/10.1145/42288.214371).
- Leith, C. E. (1975). “Climate response and fluctuation dissipation”. In: *Journal of Atmospheric Sciences* 32.10, pp. 2022–2026. DOI: [https://doi.org/10.1175/1520-0469\(1975\)032<2022:CRAFD>2.0.CO;2](https://doi.org/10.1175/1520-0469(1975)032<2022:CRAFD>2.0.CO;2).
- Leung, N., M. Abdelhafez, J. Koch, and D. Schuster (2017). “Speedup for quantum optimal control from automatic differentiation based on graphics processing units”. In: *Physical Review A* 95.4, p. 042318. DOI: [10.1103/PhysRevA.95.042318](https://doi.org/10.1103/PhysRevA.95.042318).
- Lewis, J. M. and J. C. Derber (1985). “The use of adjoint equations to solve a variational adjustment problem with advective constraints”. In: *Tellus A* 37A.4, pp. 309–322. DOI: [10.1111/j.1600-0870.1985.tb00430.x](https://doi.org/10.1111/j.1600-0870.1985.tb00430.x).
- 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, X., T.-K. L. Wong, R. T. Chen, and D. Duvenaud (2020b). “Scalable gradients for stochastic differential equations”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3870–3882.
- Lion, S. (2018). “Theoretical Approaches in Evolutionary Ecology: Environmental Feedback as a Unifying Perspective”. In: *American Naturalist* 191.1, pp. 21–44. DOI: [10.1086/694865](https://doi.org/10.1086/694865).
- Lions, J. L. (1971). *Optimal control of systems governed by partial differential equations*. Vol. 170. Springer.
- Liu, C., A. Köhl, and D. Stammer (2012). “Adjoint-Based Estimation of Eddy-Induced Tracer Mixing Parameters in the Global Ocean”. In: *Journal of Physical Oceanography* 42.7, pp. 1186–1206. DOI: [10.1175/jpo-d-11-0162.1](https://doi.org/10.1175/jpo-d-11-0162.1).
- Liu, R. and L. Zhu (2023). “Specification testing for ordinary differential equation models with fixed design and applications to COVID-19 epidemic models”. In: *Computational Statistics & Data Analysis* 180, p. 107616.
- Lorenz, E. N. (1963). “Deterministic Nonperiodic Flow”. In: *Journal of the Atmospheric Sciences* 20.2, pp. 130–141. DOI: [10.1175/1520-0469\(1963\)020<0130:dnf>2.0.co;2](https://doi.org/10.1175/1520-0469(1963)020<0130:dnf>2.0.co;2).
- 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](https://doi.org/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](https://doi.org/10.1137/0704019).

- Lyu, G., A. Köhl, I. Matei, and D. Stammer (2018). “Adjoint-Based Climate Model Tuning: Application to the Planet Simulator”. In: *Journal of Advances in Modeling Earth Systems* 10.1, pp. 207–222. DOI: [10.1002/2017ms001194](https://doi.org/10.1002/2017ms001194).
- Ma, Y., V. Dixit, M. J. Innes, X. Guo, and C. Rackauckas (2021). “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.
- 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.
- 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.
- Margossian, C. C. (2019). “A review of automatic differentiation and its efficient implementation”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.4, pp. 1–19. DOI: [10.1002/widm.1305](https://doi.org/10.1002/widm.1305).
- Marotzke, J., R. Giering, K. Q. Zhang, D. Stammer, C. Hill, and T. Lee (1999). “Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity”. In: *Journal of Geophysical Research: Oceans* 104.C12, pp. 29529–29547. DOI: [10.1029/1999jc900236](https://doi.org/10.1029/1999jc900236).
- 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](https://doi.org/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.
- Mathur, R. (2012). “An analytical approach to computing step sizes for finite-difference derivatives”. PhD thesis.
- 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](https://doi.org/10.1088/1741-4326/abcd76).
- Meehl, G. A., J. H. Richter, H. Teng, A. Capotondi, K. Cobb, F. Doblas-Reyes, M. G. Donat, M. H. England, J. C. Fyfe, W. Han, H. Kim, B. P. Kirtman, Y. Kushnir, N. S. Lovenduski, M. E. Mann, W. J. Merryfield, V. Nieves, K. Pegion, N. Rosenbloom, S. C. Sanchez, A. A. Scaife, D. Smith, A. C. Subramanian, L. Sun, D. Thompson, C. C. Ummenhofer, and S.-P. Xie (2021). “Initialized Earth System prediction from subseasonal to decadal timescales”. In: *Nature Reviews Earth & Environment*, pp. 1–18. DOI: [10.1038/s43017-021-00155-x](https://doi.org/10.1038/s43017-021-00155-x).
- Meuwly, M. (2021). “Machine learning for chemical reactions”. In: *Chemical Reviews* 121.16, pp. 10218–10239.
- Min, S., N. Kikuchi, Y. Park, S. Kim, and S. Chang (1999). “Optimal topology design of structures under dynamic loads”. In: *Structural optimization* 17, pp. 208–218.
- 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](https://doi.org/10.21105/joss.01292).
- 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](https://doi.org/10.1146/annurev.fluid.36.050802.121926).
- (2009). *Applied shape optimization for fluids*. OUP Oxford.
- Molesky, S., Z. Lin, A. Y. Piggott, W. Jin, J. Vucković, and A. W. Rodriguez (2018). “Inverse design in nanophotonics”. In: *Nature Photonics* 12.11, pp. 659–670. DOI: [10.1038/s41566-018-0246-9](https://doi.org/10.1038/s41566-018-0246-9).
- Montoison, A. and D. Orban (2023). “Krylov.JL: A Julia Basket of Hand-Picked Krylovmethods”. In: *Journal of Open Source Software* 8.89, p. 5187. DOI: [10.21105/joss.05187](https://doi.org/10.21105/joss.05187).
- Moore, A. M. and R. Kleeman (1997a). “The singular vectors of a coupled ocean-atmosphere model of Enso. I: Thermodynamics, energetics and error growth”. In: *Quarterly Journal of the Royal Meteorological Society* 123.540, pp. 953–981. DOI: [10.1002/qj.49712354009](https://doi.org/10.1002/qj.49712354009).
- (1997b). “The singular vectors of a coupled ocean-atmosphere model of Enso. II: Sensitivity studies and dynamical interpretation”. In: *Quarterly Journal of the Royal Meteorological Society* 123.540, pp. 983–1006. DOI: [10.1002/qj.49712354010](https://doi.org/10.1002/qj.49712354010).
- Moore, A. M., H. G. Arango, G. Broquet, B. S. Powell, A. T. Weaver, and J. Zavala-Garay (2011). “The Regional Ocean Modeling System (ROMS) 4-dimensional variational data assimilation systems: Part I -

- System overview and formulation”. In: *Progress in Oceanography* 91.1, pp. 34–49. DOI: [10.1016/j.pcean.2011.05.004](https://doi.org/10.1016/j.pcean.2011.05.004).
- Moore, A. M., H. G. Arango, E. D. Lorenzo, B. D. Cornuelle, A. J. Miller, and D. J. Neilson (2004). “A comprehensive ocean prediction and analysis system based on the tangent linear and adjoint of a regional ocean model”. In: *Ocean Modelling* 7.1-2, pp. 227–258. DOI: [10.1016/j.ocemod.2003.11.001](https://doi.org/10.1016/j.ocemod.2003.11.001).
- Morlighem, M., H. Seroussi, E. Larour, and E. Rignot (2013). “Inversion of basal friction in Antarctica using exact and incomplete adjoints of a higher-order model”. In: *Journal of Geophysical Research: Earth Surface* 118.3, pp. 1746–1753.
- Morlighem, M. and D. Goldberg (2023). “Data Assimilation in Glaciology”. In: pp. 93–111. DOI: [10.1017/9781009180412.007](https://doi.org/10.1017/9781009180412.007).
- Mosbeux, C., F. Gillet-Chaulet, and O. Gagliardini (2016). “Comparison of adjoint and nudging methods to initialise ice sheet model basal conditions”. In: *Geoscientific Model Development* 9.7, pp. 2549–2562.
- 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: *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis* 00, pp. 1–18. DOI: [10.1145/3458817.3476165](https://doi.org/10.1145/3458817.3476165).
- Muehlebach, M. and M. I. Jordan (2021). “Optimization with Momentum: Dynamical, Control-Theoretic, and Symplectic Perspectives”. In: *Journal of Machine Learning Research* 22.73, pp. 1–50.
- Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.
- Nadarajah, S. and A. Jameson (2000). “A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization”. In: *38th Aerospace sciences meeting and exhibit*, p. 667.
- Naumann, U. (2011). *The Art of Differentiating Computer Programs*. Society for Industrial and Applied Mathematics. DOI: [10.1137/1.9781611972078](https://doi.org/10.1137/1.9781611972078).
- Neal, R. M. et al. (2011). “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11, p. 2.
- 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](https://doi.org/10.48550/arxiv.1801.03614).
- Ni, A. (2020). “Fast linear response algorithm for differentiating chaos”. In: *arXiv preprint arXiv:2009.00595*. DOI: <https://arxiv.org/abs/2009.00595v5>.
- Ni, A. and C. Talnikar (2019a). “Adjoint sensitivity analysis on chaotic dynamical systems by Non-Intrusive Least Squares Adjoint Shadowing (NILSAS)”. In: *Journal of Computational Physics* 395, pp. 690–709. DOI: <https://doi.org/10.1016/j.jcp.2019.06.035>.
- Ni, A. and Q. Wang (2017). “Sensitivity analysis on chaotic dynamical systems by Non-Intrusive Least Squares Shadowing (NILSS)”. In: *Journal of Computational Physics* 347, pp. 56–77. DOI: <https://doi.org/10.1016/j.jcp.2017.06.033>.
- Ni, A., Q. Wang, P. Fernandez, and C. Talnikar (2019b). “Sensitivity analysis on chaotic dynamical systems by finite difference non-intrusive least squares shadowing (FD-NILSS)”. In: *Journal of Computational Physics* 394, pp. 615–631. DOI: <https://doi.org/10.1016/j.jcp.2019.06.004>.
- Nocedal, J. and S. J. Wright (1999). *Numerical optimization*. Springer.
- Norcliffe, A. and M. P. Deisenroth (2023). “Faster Training of Neural ODEs Using Gauß-Legendre Quadrature”. In: *arXiv*. DOI: [10.48550/arxiv.2308.10644](https://doi.org/10.48550/arxiv.2308.10644).
- Nurbekyan, L., W. Lei, and Y. Yang (2023). “Efficient Natural Gradient Descent Methods for Large-Scale PDE-Based Optimization Problems”. In: *SIAM Journal on Scientific Computing* 45.4, A1621–A1655. DOI: [10.1137/22M1477805](https://doi.org/10.1137/22M1477805).
- Oktay, D., N. McGreivy, J. Aduol, A. Beatson, and R. P. Adams (2020). “Randomized Automatic Differentiation”. In: *arXiv*. DOI: [10.48550/arxiv.2007.10412](https://doi.org/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](https://doi.org/10.48550/arxiv.2005.13420).
- Othmer, C. (2014). “Adjoint methods for car aerodynamics”. In: *Journal of Mathematics in Industry* 4.1, p. 6. DOI: [10.1186/2190-5983-4-6](https://doi.org/10.1186/2190-5983-4-6).

- Page, K. M. and M. A. Nowak (Nov. 2002). “Unifying Evolutionary Dynamics”. In: *Journal of Theoretical Biology* 219.1, pp. 93–98. DOI: [10.1006/jtbi.2002.3112](https://doi.org/10.1006/jtbi.2002.3112).
- Pal, A., F. Holtorf, A. Larsson, T. Loman, F. Schaefer, Q. Qu, A. Edelman, C. Rackauckas, et al. (2024). “NonlinearSolve.jl: High-Performance and Robust Solvers for Systems of Nonlinear Equations in Julia”. In: *arXiv preprint arXiv:2403.16341*.
- Pal, A., Y. Ma, V. Shah, and C. V. Rackauckas (2021). “Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics”. In: *International Conference on Machine Learning*. PMLR, pp. 8325–8335.
- Palmer, T. N., R. Buizza, F. Molteni, Y.-Q. Chen, and S. Corti (1994). “Singular vectors and the predictability of weather and climate”. In: *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences* 348.1688, pp. 459–475. DOI: [10.1098/rsta.1994.0105](https://doi.org/10.1098/rsta.1994.0105).
- 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](https://doi.org/10.3390/math8111947).
- Pantel, J. H. and L. Becks (June 2023). “Statistical Methods to Identify Mechanisms in Studies of Eco-Evolutionary Dynamics”. In: *Trends in Ecology & Evolution*, S0169534723000800. DOI: [10.1016/j.tree.2023.03.011](https://doi.org/10.1016/j.tree.2023.03.011).
- Paredes, J. A., K. Hufkens, and B. D. Stocker (Nov. 24, 2023). *Rsofun: A Model-Data Integration Framework for Simulating Ecosystem Processes*. DOI: [10.1101/2023.11.24.568574](https://doi.org/10.1101/2023.11.24.568574). URL: <https://www.biorxiv.org/content/10.1101/2023.11.24.568574v1> (visited on 11/28/2023). preprint.
- Park, S. K., K. K. Droegemeier, and C. H. Bischof (1996). “Automatic differentiation as a tool for sensitivity analysis of a convective storm in a 3-D cloud model.pdf”. In: *Computational Differentiation: Techniques, Applications and Tools*. SIAM, pp. 75–120.
- Park, S. K. and K. K. Droegemeier (2000). “Sensitivity Analysis of a 3D Convective Storm: Implications for Variational Data Assimilation and Forecast Error”. In: *Monthly Weather Review* 128.1, pp. 140–159. DOI: [10.1175/1520-0493\(2000\)128<0140:saoacs>2.0.co;2](https://doi.org/10.1175/1520-0493(2000)128<0140:saoacs>2.0.co;2).
- Paul, D., J. Peng, and P. Burman (2011). “Semiparametric modeling of autonomous nonlinear dynamical systems with application to plant growth”. In: *The Annals of Applied Statistics* 5.3, pp. 2078–2108. DOI: [10.1214/11-AOAS459](https://doi.org/10.1214/11-AOAS459).
- Petra, N., J. Martin, G. Stadler, and O. Ghattas (2014). “A Computational Framework for Infinite-Dimensional Bayesian Inverse Problems, Part II: Stochastic Newton MCMC with Application to Ice Sheet Flow Inverse Problems”. In: *SIAM Journal on Scientific Computing* 36.4, A1525–A1555. DOI: [10.1137/130934805](https://doi.org/10.1137/130934805).
- Petra, N., H. Zhu, G. Stadler, T. J. Hughes, and O. Ghattas (2012). “An inexact Gauss–Newton method for inversion of basal sliding and rheology parameters in a nonlinear Stokes ice sheet model”. In: *Journal of Glaciology* 58.211, pp. 889–903. DOI: [10.3189/2012jog11j182](https://doi.org/10.3189/2012jog11j182).
- Pichler, M. and F. Hartig (Apr. 2023). “Machine Learning and Deep Learning—A Review for Ecologists”. In: *Methods in Ecology and Evolution* 14.4, pp. 994–1016. DOI: [10.1111/2041-210X.14061](https://doi.org/10.1111/2041-210X.14061).
- Pironneau, O. (1974). “On optimum design in fluid mechanics”. In: *Journal of Fluid Mechanics* 64.1, pp. 97–110. DOI: [10.1017/S0022112074002023](https://doi.org/10.1017/S0022112074002023).
- Pontarp, M., Å. Brännström, and O. L. Petchey (Apr. 2019). “Inferring Community Assembly Processes from Macroscopic Patterns Using Dynamic Eco-evolutionary Models and Approximate Bayesian Computation (ABC)”. In: *Methods in Ecology and Evolution* 10.4. Ed. by T. Poisot, pp. 450–460. DOI: [10.1111/2041-210X.13129](https://doi.org/10.1111/2041-210X.13129).
- Rabier, F., H. Järvinen, E. Klinker, J. F. Mahfouf, and A. Simmons (2000). “The ECMWF operational implementation of four-dimensional variational assimilation. I: Experimental results with simplified physics”. In: *Quarterly Journal of the Royal Meteorological Society* 126.564, pp. 1143–1170. DOI: [10.1002/qj.49712656415](https://doi.org/10.1002/qj.49712656415).
- Rabier, F. and P. Courtier (1992). “Four-Dimensional Assimilation In the Presence of Baroclinic Instability”. In: *Quarterly Journal of the Royal Meteorological Society* 118.506, pp. 649–672. DOI: [10.1002/qj.49711850604](https://doi.org/10.1002/qj.49711850604).
- 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](https://doi.org/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](https://doi.org/10.1016/j.jcp.2018.10.045).
- Ramsay, J. and G. Hooker (2017). *Dynamic Data Analysis, Modeling Data with Differential Equations*. DOI: [10.1007/978-1-4939-7190-9](https://doi.org/10.1007/978-1-4939-7190-9).
- Ramsay, J. O. (1996). “Principal differential analysis: Data reduction by differential operators”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.3, pp. 495–508.
- Ramsay, J. O., G. Hooker, D. Campbell, and J. Cao (2007). “Parameter estimation for differential equations: a generalized smoothing approach”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 69.5, pp. 741–796.
- Ramsundar, B., D. Krishnamurthy, and V. Viswanathan (2021). “Differentiable Physics: A Position Piece”. In: *arXiv*. DOI: [10.48550/arxiv.2109.07573](https://doi.org/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](https://doi.org/10.1007/s42967-021-00159-w).
- Rasp, S., M. S. Pritchard, and P. Gentile (Sept. 25, 2018). “Deep Learning to Represent Subgrid Processes in Climate Models”. In: *Proceedings of the National Academy of Sciences* 115.39, pp. 9684–9689. DOI: [10.1073/pnas.1810286115](https://doi.org/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](https://doi.org/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]*.
- 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](https://doi.org/10.1137/16m1096840).
- Ruder, S. (2016). “An overview of gradient descent optimization algorithms”. In: *arXiv*. DOI: [10.48550/arXiv.1609.04747](https://doi.org/10.48550/arXiv.1609.04747).
- Ruelle, D. (1997). “Differentiation of SRB states”. In: *Communications in Mathematical Physics* 187.1, pp. 227–241. DOI: <https://doi.org/10.1007/s002200050134>.
- (2009). “A review of linear response theory for general differentiable dynamical systems”. In: *Nonlinearity* 22.4, p. 855. DOI: <https://iopscience.iop.org/article/10.1088/0951-7715/22/4/009>.
- Ruppert, K. M., R. J. Kline, and M. S. Rahman (Jan. 2019). “Past, Present, and Future Perspectives of Environmental DNA (eDNA) Metabarcoding: A Systematic Review in Methods, Monitoring, and Applications of Global eDNA”. In: *Global Ecology and Conservation* 17, e00547. DOI: [10.1016/j.gecco.2019.e00547](https://doi.org/10.1016/j.gecco.2019.e00547).
- 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.
- Schäfer, F., M. Kloc, C. Bruder, and N. Lörch (2020). “A differentiable programming method for quantum control”. In: *Machine Learning: Science and Technology* 1.3, p. 035009. DOI: [10.1088/2632-2153/ab9802](https://doi.org/10.1088/2632-2153/ab9802).

- Schäfer, F., P. Sekatski, M. Koppenhöfer, C. Bruder, and M. Kloc (2021a). “Control of stochastic quantum dynamics by differentiable programming”. In: *Machine Learning: Science and Technology* 2.3, p. 035004. DOI: [10.1088/2632-2153/abec22](https://doi.org/10.1088/2632-2153/abec22).
- Schäfer, F., M. Tarek, L. White, and C. Rackauckas (2021b). “AbstractDifferentiation.jl: Backend-Agnostic Differentiable Programming in Julia”. In: *arXiv*. DOI: [10.48550/arxiv.2109.12449](https://doi.org/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.
- Schartau, M., P. Wallhead, J. Hemmings, U. Löptien, I. Kriest, S. Krishna, B. A. Ward, T. Slawig, and A. Oschlies (Mar. 29, 2017). “Reviews and Syntheses: Parameter Identification in Marine Planktonic Ecosystem Modelling”. In: *Biogeosciences* 14.6, pp. 1647–1701. DOI: [10.5194/bg-14-1647-2017](https://doi.org/10.5194/bg-14-1647-2017).
- 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. Applling, 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](https://doi.org/10.1038/s43017-023-00450-9).
- 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](https://doi.org/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:
- Squire, W. and G. Trapp (1998). “Using Complex Variables to Estimate Derivatives of Real Functions”. In: 40, pp. 110–112. DOI: [10.1137/s003614459631241x](https://doi.org/10.1137/s003614459631241x).
- Stammer, D., A. Köhl, A. Vlasenko, I. Matei, F. Lunkeit, and S. Schubert (2018). “A Pilot Climate Sensitivity Study Using the CEN Coupled Adjoint Model (CESAM)”. In: *Journal of Climate* 31.5, pp. 2031–2056. DOI: [10.1175/jcli-d-17-0183.1](https://doi.org/10.1175/jcli-d-17-0183.1).
- Stammer, D., C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. N. Hill, and J. Marshall (2002). “Global ocean circulation during 1992–1997, estimated from ocean observations and a general circulation model”. In: *Journal of Geophysical Research: Oceans* 107.C9, pp. 1–1-1-27. DOI: [10.1029/2001jc000888](https://doi.org/10.1029/2001jc000888).
- Stammer, D. (2005). “Adjusting Internal Model Errors through Ocean State Estimation”. In: *Journal of Physical Oceanography* 35.6, pp. 1143–1153. DOI: [10.1175/jpo2733.1](https://doi.org/10.1175/jpo2733.1).
- Stein, E. M. and R. Shakarchi (2010). *Complex analysis*. Vol. 2. Princeton University Press.
- Strogatz, S. H. (2018). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press.
- Tadmor, E. (2012). “A review of numerical methods for nonlinear partial differential equations”. In: *Bulletin of the American Mathematical Society* 49.4, pp. 507–554.
- Talagrand, O. and P. Courtier (1987). “Variational Assimilation of Meteorological Observations With the Adjoint Vorticity Equation. I: Theory”. In: *Quarterly Journal of the Royal Meteorological Society* 113.478, pp. 1311–1328. DOI: [10.1002/qj.49711347812](https://doi.org/10.1002/qj.49711347812).
- Tarantola, A. (2007). “Mapping of Probabilities”. In: *Book*, pp. 1–305.
- Thacker, W. C. (1988). “Fitting models to inadequate data by enforcing spatial and temporal smoothness”. In: *Journal of Geophysical Research: Oceans (1978–2012)* 93.C9, pp. 10655–10665. DOI: [10.1029/jc093ic09p10655](https://doi.org/10.1029/jc093ic09p10655).
- (1989). “The role of the Hessian matrix in fitting models to measurements”. In: *Journal of Geophysical Research: Oceans (1978–2012)* 94.C5, pp. 6177–6196. DOI: [10.1029/jc094ic05p06177](https://doi.org/10.1029/jc094ic05p06177).
- Thacker, W. C. and R. B. Long (1988). “Fitting dynamics to data”. In: *Journal of Geophysical Research: Oceans (1978–2012)* 93.C2, pp. 1227–1240. DOI: [10.1029/jc093ic02p01227](https://doi.org/10.1029/jc093ic02p01227).

- Thompson, J. F., B. K. Soni, and N. P. Weatherill (1998). *Handbook of grid generation*. CRC press.
- Thuburn, J. (2005). “Climate sensitivities via a Fokker–Planck adjoint approach”. In: *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography* 131.605, pp. 73–92. DOI: <https://doi.org/10.1256/qj.04.46>.
- Thuerey, N., P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um (2021). *Physics-based Deep Learning*. WWW.
- Transtrum, M. K., B. B. Machta, and J. P. Sethna (2011). “Geometry of nonlinear least squares with applications to sloppy models and optimization”. In: *Physical Review E* 83.3, p. 036701.
- 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](https://doi.org/10.1016/j.camwa.2011.06.002).
- Tziperman, E., W. C. Thacker, and R. B. Long (1992a). “Oceanic data analysis using a general circulation model. Part I: Simulations”. In: *Journal of Physical Oceanography* 22.12, pp. 1434–1457. DOI: [10.1175/1520-0485\(1992\)022<1434:odauag>2.0.co;2](https://doi.org/10.1175/1520-0485(1992)022<1434:odauag>2.0.co;2).
- (1992b). “Oceanic data analysis using a general circulation model. Part II: A North Atlantic model”. In: *Journal of Physical Oceanography* 22.12, pp. 1458–1485. DOI: [10.1175/1520-0485\(1992\)022<1458:odauag>2.0.co;2](https://doi.org/10.1175/1520-0485(1992)022<1458:odauag>2.0.co;2).
- Tziperman, E. and W. C. Thacker (1989). “An Optimal-Control/Adjoint-Equations Approach to Studying the Oceanic General Circulation”. In: *Journal of Physical Oceanography* 19.10, pp. 1471–1485. DOI: [10.1175/1520-0485\(1989\)019<1471:aoceat>2.0.co;2](https://doi.org/10.1175/1520-0485(1989)019<1471:aoceat>2.0.co;2).
- Utke, J., U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch (2008). “OpenAD/F: A Modular Open-Source Tool for Automatic Differentiation of Fortran Codes”. In: *ACM Transactions on Mathematical Software (TOMS)* 34.4, p. 18. DOI: [10.1145/1377596.1377598](https://doi.org/10.1145/1377596.1377598).
- Vadlamani, S. K., T. P. Xiao, and E. Yablonovitch (2020). “Physics successfully implements Lagrange multiplier optimization”. In: *Proceedings of the National Academy of Sciences* 117.43, pp. 26639–26650. DOI: [10.1073/pnas.2015192117](https://doi.org/10.1073/pnas.2015192117).
- Vadyala, S. R., S. N. Betgeri, J. C. Matthews, and E. Matthews (2022). “A review of physics-based machine learning in civil engineering”. In: *Results in Engineering* 13, p. 100316.
- Vallis, G. K. (2016). “Geophysical fluid dynamics: whence, whither and why?” In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 472.2192, pp. 20160140–23. DOI: [10.1098/rspa.2016.0140](https://doi.org/10.1098/rspa.2016.0140).
- Van den Berg, N. I., D. Machado, S. Santos, I. Rocha, J. Chacón, W. Harcombe, S. Mitri, and K. R. Patil (May 16, 2022). “Ecological Modelling Approaches for Predicting Emergent Properties in Microbial Communities”. In: *Nature Ecology & Evolution*. DOI: [10.1038/s41559-022-01746-7](https://doi.org/10.1038/s41559-022-01746-7).
- Van Keulen, F., R. Haftka, and N.-H. Kim (2005). “Review of options for structural design sensitivity analysis. Part 1: Linear systems”. In: *Computer methods in applied mechanics and engineering* 194.30–33, pp. 3213–3243.
- Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc.
- Vidard, A., P. A. Bouttier, and F. Vigilant (2015). “NEMOTAM: tangent and adjoint models for the ocean modelling platform NEMO”. In: *Geoscientific Model Development* 8.4, pp. 1245–1257. DOI: [10.5194/gmd-8-1245-2015](https://doi.org/10.5194/gmd-8-1245-2015).
- Vieli, A., A. J. Payne, Z. Du, and A. Shepherd (2006). “Numerical modelling and data assimilation of the Larsen B ice shelf, Antarctic Peninsula”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 364.1844, pp. 1815–1839. DOI: [10.1098/rsta.2006.1800](https://doi.org/10.1098/rsta.2006.1800).
- Villa, C., M. A. J. Chaplain, and T. Lorenzi (Mar. 6, 2021). “Evolutionary Dynamics in Vascularised Tumours under Chemotherapy: Mathematical Modelling, Asymptotic Analysis and Numerical Simulations”. In: *Vietnam Journal of Mathematics* 49.1, pp. 143–167. DOI: [10.1007/s10013-020-00445-9](https://doi.org/10.1007/s10013-020-00445-9).

- 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](https://doi.org/10.1007/s10589-006-0397-3).
- Walther, A. and A. Griewank (2004). “Advantages of Binomial Checkpointing for Memory-reduced Adjoint Calculations”. In: *Proceedings of ENUMATH 2003 the 5th European Conference on Numerical Mathematics and Advanced Applications Prague, August 2003*, pp. 834–843. DOI: [10.1007/978-3-642-18775-9_82](https://doi.org/10.1007/978-3-642-18775-9_82).
- Wang, F., D. Zheng, J. Decker, X. Wu, G. M. Essertel, and T. Rompf (2019). “Backpropagation with Continuation Callbacks: Foundations for Efficient and Expressive Differentiable Programming”. In: *Proceedings of the ACM on Programming Languages* 3.ICFP, p. 96. DOI: [10.1145/3341700](https://doi.org/10.1145/3341700).
- Wang, Q. (2013). “Forward and adjoint sensitivity computation of chaotic dynamical systems”. In: *Journal of Computational Physics* 235, pp. 1–13. DOI: <https://doi.org/10.1016/j.jcp.2012.09.007>.
- (2014). “Convergence of the least squares shadowing method for computing derivative of ergodic averages”. In: *SIAM Journal on Numerical Analysis* 52.1, pp. 156–170. DOI: <https://doi.org/10.1137/130917065>.
- Wang, Q., R. Hu, and P. Blonigan (2014a). “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>.
- (2014b). “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, and C. Cowen-Breen (2022). “Discovering the rheology of Antarctic Ice Shelves via physics-informed deep learning”. In: *Journal of Computational Physics* 448, pp. 110815–110830. DOI: <https://doi.org/10.1016/j.jcp.2021.110815>.
- Wanner, G. and E. Hairer (1996). *Solving ordinary differential equations II*. Vol. 375. Springer Berlin Heidelberg New York.
- Watts, M. C. (Oct. 1, 2001). “Modelling and the Monitoring of Mesocosm Experiments: Two Case Studies”. In: *Journal of Plankton Research* 23.10, pp. 1081–1093. DOI: [10.1093/plankt/23.10.1081](https://doi.org/10.1093/plankt/23.10.1081).
- Weaver, A. T., J. Vialard, and D. L. T. Anderson (2003). “Three- and Four-Dimensional Variational Assimilation with a General Circulation Model of the Tropical Pacific Ocean. Part I: Formulation, Internal Diagnostics, and Consistency Checks”. In: *Monthly Weather Review* 131.7, pp. 1360–1378. DOI: [10.1175/1520-0493\(2003\)131<1360:tafvaw>2.0.co;2](https://doi.org/10.1175/1520-0493(2003)131<1360:tafvaw>2.0.co;2).
- Weng, E. S., S. Malyshev, J. W. Lichstein, C. E. Farrior, R. Dybzinski, T. Zhang, E. Shevliakova, and S. W. Pacala (May 7, 2015). “Scaling from Individual Trees to Forests in an Earth System Modeling Framework Using a Mathematically Tractable Model of Height-Structured Competition”. In: *Biogeosciences* 12.9, pp. 2655–2694. DOI: [10.5194/bg-12-2655-2015](https://doi.org/10.5194/bg-12-2655-2015).
- 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](https://doi.org/10.1145/355586.364791).
- 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](https://doi.org/10.1145/356012.356013).
- Wunsch, C. (2008). “The circulation of the ocean and its variability”. In: *Progress in Physical Geography* 32.4, pp. 463–474. DOI: [10.1177/0309133308096753](https://doi.org/10.1177/0309133308096753).
- Wunsch, C. and P. Heimbach (2007). “Practical global oceanic state estimation”. In: *Physica D: Nonlinear Phenomena* 230.1-2, pp. 197–208. DOI: [10.1016/j.physd.2006.09.040](https://doi.org/10.1016/j.physd.2006.09.040).
- Xu, P., F. Roosta, and M. W. Mahoney (n.d.). “Second-order Optimization for Non-convex Machine Learning: an Empirical Study”. In: *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, pp. 199–207. DOI: [10.1137/1.9781611976236.23](https://doi.org/10.1137/1.9781611976236.23).
- Yazdani, A., L. Lu, M. Raissi, and G. E. Karniadakis (Nov. 18, 2020). “Systems Biology Informed Deep Learning for Inferring Parameters and Hidden Dynamics”. In: *PLOS Computational Biology* 16.11. Ed. by V. Hatzimanikatis, e1007575. DOI: [10.1371/journal.pcbi.1007575](https://doi.org/10.1371/journal.pcbi.1007575).
- Zanna, L., P. Heimbach, A. M. Moore, and E. Tziperman (2012). “Upper-ocean singular vectors of the North Atlantic climate with implications for linear predictability and variability”. In: *Quarterly Journal of the Royal Meteorological Society* 138.663, pp. 500–513. DOI: [10.1002/qj.937](https://doi.org/10.1002/qj.937).

- Zanna, L., P. Heimbach, A. M. Moore, and E. Tziperman (2011). “Optimal Excitation of Interannual Atlantic Meridional Overturning Circulation Variability”. In: *Journal of Climate* 24.2, pp. 413–427. DOI: [10.1175/2010jcli3610.1](https://doi.org/10.1175/2010jcli3610.1).
- (2010). “The Role of Ocean Dynamics in the Optimal Growth of Tropical SST Anomalies”. In: *Journal of Physical Oceanography* 40.5, pp. 983–1003. DOI: [10.1175/2009jpo4196.1](https://doi.org/10.1175/2009jpo4196.1).
- Zdeborová, L. (2020). “Understanding deep learning is also a job for physicists”. en. In: *Nature Physics*. DOI: [10.1038/s41567-020-0929-2](https://doi.org/10.1038/s41567-020-0929-2).
- 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.
- Zhang, X., J. Cao, and R. J. Carroll (2015). “On the selection of ordinary differential equation models with application to predator-prey dynamical models”. In: *Biometrics* 71.1, pp. 131–138.
- Zhu, W., K. Xu, E. Darve, and G. C. Beroza (2021). “A general approach to seismic inversion with automatic differentiation”. In: *Computers & Geosciences* 151, p. 104751. DOI: [10.1016/j.cageo.2021.104751](https://doi.org/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](https://doi.org/10.48550/arxiv.2107.09443).