A Short Review of Automatic Differentiation Pitfalls in Scientific Computing

Jan Hückelheim [1] **Harshitha Menon** [2] **William Moses** [3] **Bruce Christianson** [4] **Paul Hovland** [1] **Laurent Hascoët** [5]

Abstract

Automatic differentiation, also known as backpropagation, AD, autodiff, or algorithmic differentiation, is a popular technique for computing derivatives of computer programs. While AD has been successfully used in countless engineering, science and machine learning applications, it can sometimes nevertheless produce surprising results. In this paper we categorize problematic usages of AD and illustrate each category with examples such as chaos, time-averages, discretizations, fixed-point loops, lookup tables, linear solvers, and probabilistic programs, in the hope that readers may more easily avoid or detect such pitfalls.

1. Introduction

Automatic differentiation (AD) is steadily becoming more popular in machine learning, scientific computing, engineering, and many other fields as a tool to compute derivatives efficiently and accurately. While the benefits are widely popularized, users are not always aware that AD can produce surprising results when applied to certain functions. Some of the most problematic failure modes are inherent to common AD approaches and systematically lead to confusing (or, by some measure, incorrect) derivatives across multiple tools or input languages. This makes debugging difficult without an in-depth understanding.

This article presents a new way to categorize AD problems from previous work (Fischer, 1991; Beck & Fischer, 1994; Griewank & Walther, 2008; Christianson, 1994; Bangaru et al., 2021), with the goal of enabling readers of this article to avoid known pitfalls, and be less surprised when they discover new ones in the future.

[1]Argonne National Laboratory, 9700 S. Cass Ave., Lemont, IL 60439, USA [2]Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, CA 94551, USA [3]MIT CSAIL, 32 Vassar St, Cambridge, MA 02139, USA [4]University of Hertfordshire, College Lane, Hatfield, AL10 9AB, UK [5]Inria, 2004 Rte des Lucioles, 06902 Valbonne, France. Correspondence to: Jan Hückelheim <jhueckelheim@anl.gov>.

Published at the Differentiable Almost Everything Workshop of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023. Copyright 2023 by the author(s).

To define the desired outcome of AD, let us consider a mathematical function $y \leftarrow f(x)$, where x, y are real or complex scalar values or vectors of arbitrary size. One might wish to compute $\nabla(f)$, the derivative of y with respect to x, which—depending on the shape and size of x and y—could be a scalar, column or row vector, or, more generally, a Jacobian matrix. Instead of computing an entire Jacobian matrix, one might wish to compute projections of that matrix (popularly implemented as the forward mode of automatic differentiation) or of its transpose (implemented as reverse mode or backpropagation). AD can also compute higher-order derivatives or their projections. The pitfalls described in this work apply to all of these AD usages; we refer to prior work that discusses AD modes and their relationship (Giering & Kaminski, 1998; Bartholomew-Biggs et al., 2000; Griewank, 2003; Griewank & Walther, 2008; Naumann, 2012; Hoffmann, 2016; Baydin et al., 2017; Margossian, 2019; Gebremedhin & Walther, 2020; Radul et al., 2023).

Instead of the *true* mathematical function f, a computer program implements an approximation $Y \leftarrow F(X)$, where F, X, and Y differ from the true f, x, and y for reasons that include floating-point errors and often many other approximations. Since AD operates on the computer program, we can at best hope to obtain derivatives of F. Moreover, AD is applied at some level of abstraction, which influences the exact function F that we consider the program to implement. We can thus define the expected output of AD as follows:

> *AD operates on a chosen abstraction level and assuming that operations and their derivatives can be computed to sufficient accuracy. Under these assumptions, AD computes the derivative of a function F given as the composition of operations encountered during the evaluation of a particular branch of a program above the given abstraction level.*

This definition allows us to categorize pitfalls in Section 2. Before doing so, we note that AD approaches have various trade-offs regarding performance, user convenience, and tool development effort (Hascoët & Utke, 2016; Margossian, 2019). On these grounds, some authors argue in favor of AD on high abstraction levels (Farrell et al., 2013) while others argue for the opposite (Moses & Churavy, 2020). As important as these aspects are, we focus on the semantics of AD and also ignore problems caused by tool bugs.

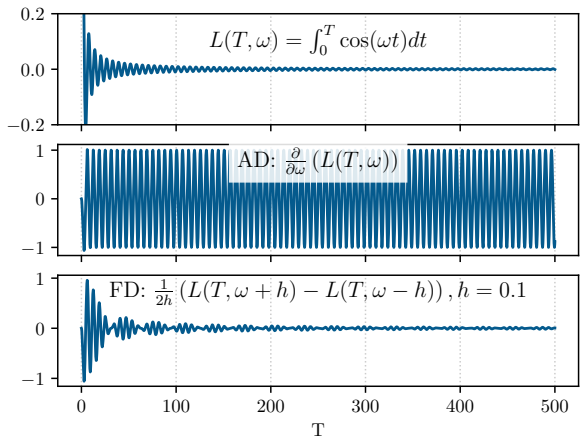

Figure 1. A function with unintuitive derivatives. **Top:** The time average converges to zero, regardless of the frequency. One might thus expect the derivative with respect to the frequency parameter to converge to zero. **Middle:** The derivative, however, continues to oscillate between -1 and 1. **Bottom:** Finite differences (FD) behave more intuitively and converge to zero.

2. Categories of AD pitfalls

In this section we illustrate each pitfall category with examples and refer to related work for more details.

2.1. Pitfall I: Unexpected function derivatives

The derivatives of the true function f do not always exist or are not always useful. For example, a program may model discontinuous physical processes such as phase changes from liquid to solid, or shocks in fluid dynamics. Sometimes the discontinuities are introduced when modeling smooth processes, for example when quantities are filtered to avoid nonphysical behavior or numerical instabilities (Sweby, 1984). Discretized models are often designed as good approximations of a continuous system, but their derivatives might be vastly different (Hager, 2000; Collis et al.). In some cases the derivatives of the discretized system are useful, and in others a discretization of a differentiated system is preferable (Nadarajah & Jameson, 2000). The computational fluid dynamics community–being an early user of AD and adjoint methods–has studied this in detail (Appel & Gunzburger, 1997; Fikl et al., 2016; Giles & Ulbrich, 2010; Bardos & Pironneau, 2002), but practitioners in other domains may find similar problems when they adopt AD.

Differentiable functions with well-understood large-scale behavior may have localized behavior that causes unintuitive, noisy, or misleading derivatives. Previous work has found that finite differences can in such cases be more reliable (Moré & Wild, 2014; Dussault & Hamelin, 2006). One example is the computation of time averages or other statistical properties of dynamic or chaotic processes, which can have exponentially diverging derivatives (Wang, 2013) for finite-size time windows. From a user's perspective, it can be difficult to determine whether a program (or a small part of it) implements a chaotic function, which may appear as an unremarkable composition of differentiable operations. Consequently, a user may become aware of small-scale chaotic effects only when applying AD. Previous work proposed methods to obtain meaningful derivatives in the presence of chaos (Wang, 2013; Wang et al., 2014; Blonigan, 2017), which go beyond the standard definition of AD and need to be used judiciously because of their high cost.

A poorly chosen objective function can itself cause misleading derivatives. Consider a system whose behavior in time is modeled as a cosine function with frequency ω. Its time-averaged state can be approximated by integrating over a sufficiently long time window $[0, T]$. While the integral converges with growing T regardless of ω (which intuitively indicates a 0-derivative with respect to ω), the derivative never converges and oscillates between -1 and 1 as shown in Figure 1. Confusingly, finite differences for any fixed h converge to the intuitive, but incorrect, value of 0. The example may appear contrived but represents a challenge in real applications, such as oscillations that cause misleading derivatives for time-averaged quantities in aerospace engineering (Krakos et al., 2012).

Undesirable derivatives are not limited to simulations of physical systems. For example, machine learning models may have "exploding" or "vanishing" gradients that grow or shrink exponentially with the number of layers (Pascanu et al., 2012; Hochreiter, 1998). In summary, we observe that useful functions, even differentiable ones, do not always have useful derivatives. The successful use of AD can require a deep understanding of the problem, and objective functions designed with differentiation in mind.

2.2. Pitfall II: Unexpected abstraction derivatives

Programs are often written in a relatively high-level abstraction and successively transformed, or *lowered*, into machine code through the use of general-purpose or domain-specific compilers, runtime calls to libraries, or a mix of multiple approaches. This is true for programs written in high-level frameworks such as Jax (Bradbury et al., 2018) or PyTorch (Paszke et al., 2017), or relatively low-level languages such as C or Fortran. Transformations are designed to be sufficiently accurate—sometimes even exact—but may nevertheless change the derivatives. AD therefore does not necessarily differentiate "what you implement" but, rather, *what is implemented at the level of abstraction at which AD is applied*. It is generally the user's responsibility to ensure that the chosen abstraction's derivatives are reasonable approximations to those of the true function. There are almost always levels of abstraction that are too low to

permit this, such as the low-level view in which numbers are represented as bit patterns. Viewed at this level, programs have discrete inputs and outputs and are therefore non-differentiable. Because such a view is clearly unhelpful for computing derivatives, AD *usually* operates at a higher level of abstraction, where operations are assumed to represent functions with real (or complex) inputs and outputs (Bolte & Pauwels, 2020).

The most suitable abstraction for applying AD is not always explicit in the source code. For example, multiple linear algebra operations can be implemented as a sequence of loop nests within the same routine. In this case, the linear algebra view that would often be most appropriate for AD (Giles, 2008) exists only on paper. Another example is the approximate computation of an expectation by sampling a discrete random process. While the expected value may smoothly depend on model parameters, the sampling process will hide this dependency from classic AD tools such as Jax (Bradbury et al., 2018), which motivates the development of probabilistic programming approaches (Arya et al., 2023) that allow expressing such problems on a more suitable level of abstraction. Users may need to modify their implementation or even completely re-implement their problem in a different language (Bangaru et al., 2021) to expose the appropriate abstraction to an AD tool.

Explicit functions are sometimes replaced with approximations whose derivatives differ from those of the approximated function. As an example, consider table lookups that are used in practice both to approximate small math operations (Alexe et al., Dec 2009) or entire physical models or to incorporate empirical observations (Ahmed et al., 2009), but result in a piecewise constant function with zero derivatives everywhere. Similar issues occur for other approximation techniques including bit hacks (Ercegovac et al., 2000; Volder, 1959), numerical quadrature (Bangaru et al., 2021), and probabilistic methods (Arya et al., 2023). Some programs approximate the same function by using different algorithms depending on the input value (Gal, 1991), and differentiating through such an implementation yields useful derivatives for some but not all inputs.

Implicit functions include the solution of systems of linear or nonlinear equations using direct or iterative processes. Instead of differentiating through a linear solver implementation, it is usually preferable to formulate the derivative on a higher abstraction level in terms of a modified linear equation system (Griewank et al., 1993; Giering & Kaminski, 1998; Bartholomew-Biggs et al., 2000; Moré & Wild, 2014). Similarly, programs containing iterative fixed-point loops often require AD at a high abstraction level because the process may terminate after a few steps (Gilbert, 1992; Beck, 1994; Christianson, 1994; 1998), especially when given a good initial guess, while the derivatives may not have converged yet. Figure 2 shows such an example. Even when addressed by AD tools, solutions to this pitfall require user awareness (Taftaf et al., 2015). Similar problems exist in deep learning (Liao et al., 2018; Bai et al., 2019; Lorraine et al., 2020), and also require special derivative rules that effectively raise the level of abstraction.

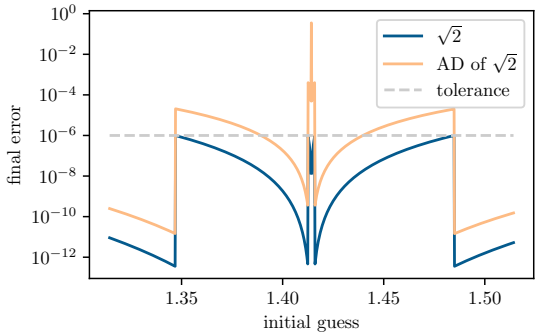

Figure 2. Square root finding with Heron's method for a set error tolerance of 10^{-6}. When differentiating through the iterative method, the stopping criterion of the primal is still used and stops the process before the derivatives reach the desired accuracy.

Without branch	With branch

Function:
`def f(x):`
`  return x`
	`def g(x):`
	`  if x == 0:`
	`  return 0`
	`  else:`
	`  return x`

Derivative:
`def f_d(x, xd):`
`  return xd`
	`def g_d(x, xd):`
	`  if x == 0:`
	`  return 0 # wrong!`
	`  else:`
	`  return xd`

Figure 3. The functions f and g return the exact same value for all inputs and implement the same mathematical function without any approximations. However, their derivative functions $\dot{f}$ and $\dot{g}$ compute different values at $x = 0$ because the encountered operators inside the branch do not depend on x.

2.3. Pitfall III: Unexpected branch derivatives

Chain rule differentiation of the operations encountered inside a branch may yield different derivatives from those encountered on other branches of the function, which is problematic if program branches apply on closed subdomains—particularly for individual points (Griewank & Walther, 2008; Beck & Fischer, 1994). For example, BLAS functions (Blackford et al., 2002) often use a fast branch if one of the factors in a multiplication is 0 or 1, resulting in zero derivatives with respect to that factor. An example is shown in Figure 3. A similar problem occurs for `max` (or `min`) functions with inputs that contain more than one identical maximum (or minimum) value, where the choice does not affect the function result but may affect the derivative.

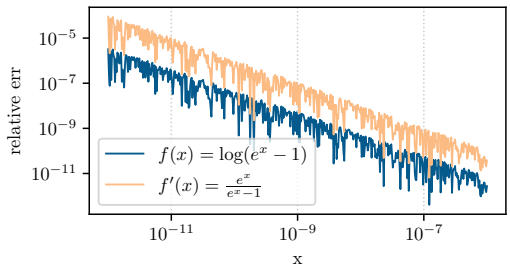

Figure 4. Example of a function whose derivative is orders of magnitude less accurate when evaluated with double precision floating-point numbers, particularly for input values close to 0.

2.4. Pitfall IV: Inaccurate Operators

Sometimes, calculating the derivative of a function can incur more severe floating-point rounding errors than can calculating the function itself. Figure 4 shows such an example, where the derivative calculation involves subtracting two nearly equal values and subsequently dividing by the resulting small number, causing large relative errors.

Roundoff is not the only concern for this pitfall. Whenever AD is applied at a high level of abstraction, differentiation occurs with the assumption that the operators exactly compute the differentiable function and its derivative. This assumption is violated, for example, when iterative linear solvers are used that converge to a reasonably accurate value for the original function but break down, diverge, or converge more slowly for the derivative function. Commonly used iterative methods such as BiCG, BiCGSTAB, or restarted GMRES do not have known performance guarantees (Barrett et al., 1994).

3. Debugging Techniques

AD can be implemented in different modes that compute projections of Jacobian matrices or of their transpose, or higher-order derivative matrices. Finite differences (FD) can be used to obtain approximations for Jacobian projections, which can be compared with AD and allows detection of tool bugs as well as many instances of pitfalls I, II and III. However, this requires choosing a good step size and sometimes leaves users guessing whether a discrepancy is caused by floating-point or truncation errors or by AD problems (Wolfe, 1982). Higher order formulae and multiple step sizes can be used for a more thorough test that determines the convergence order of the error.

If a tool supports more than one AD mode or if multiple tools are available that support different modes, users can use simple algebraic identities to compute the same quantity in multiple ways and compare the results. While this is a useful debugging tool, pitfalls I, II and III would cause all AD modes to consistently produce the same wrong derivatives. To circumvent this problem, users can mix finite differences with other AD modes, as is done for example in the gradient testing routine in PyTorch (Paszke et al., 2017) with randomized values for x, $\dot{x}$ and $\bar{y}$, allowing direct debugging of reverse mode AD – with the aforementioned caveats regarding step size and error tolerance.

4. Conclusion

We show in this paper that AD may produce wrong or surprising results. We believe that this need not be a reason to avoid AD. Many other methods—floating-point arithmetic, optimization algorithms, machine learning, to name a few—are useful despite occasional surprises.

Further research could aim to create languages and tools that implement AD in a more predictable way. Recent work discusses AD semantics and provably correct AD but focuses on functional or domain-specific languages that are more restrictive than the languages used in most applications (Huot et al., 2020; Krawiec et al., 2022; Wang et al., 2019). There has also been recent progress in understanding the semantics of AD for neural networks containing nonsmooth activation functions or that use finite precision arithmetic (Lee et al., 2020; 2023), or for probabilistic programs (Lew et al., 2023).

Programmers are often aware that they are responsible for the semantics of their program and that the language only guarantees semantics of individual constructs. We argue that a similar view of AD should be encouraged rather than making sweeping claims about AD "differentiating programs." The quote from (Naumann, 2012) remains true a decade later: *The application of AD to computer programs still deserves to be called an "art."*

Acknowledgments

We thank Christian Bischof, Valentin Churavy, and Jesse Michel for insightful comments and for bringing pitfall examples to our attention. We are grateful for discussions with the late Andreas Griewank, who greatly influenced our work. This work was supported by the Applied Mathematics activity within the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under contract number DE-AC02-06CH11357, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, LANL grant 531711, and by the US Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator 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.

References

Ahmed, Z., Akerib, D., Arrenberg, S., Attisha, M., Bailey, C., Baudis, L., Bauer, D., Beaty, J., Brink, P., Bruch, T., et al. Search for weakly interacting massive particles with the first five-tower data from the cryogenic dark matter search at the soudan underground laboratory. *Physical Review Letters*, 102 (1):011301, 2009.

Alexe, M., Roderick, O., Utke, J., Anitescu, M., Hovland, P., and Fanning, T. Automatic differentiation of codes in nuclear engineering applications. Technical Report ANL/MCS-TM–310, United States, Dec 2009. URL http://inis.iaea.org/search/search.aspx?orig_q=RN:41049456.

Appel, J. R. and Gunzburger, M. D. Difficulties in sensitivity calculations for flows with discontinuities. *AIAA Journal*, 35 (5):842–848, 1997.

Arya, G., Schauer, M., Schäfer, F., and Rackauckas, C. Automatic differentiation of programs with discrete randomness, 2023.

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

Bangaru, S. P., Michel, J., Mu, K., Bernstein, G., Li, T.-M., and Ragan-Kelley, J. Systematically differentiating parametric discontinuities. *ACM Transactions on Graphics (TOG)*, 40(4): 1–18, 2021.

Bardos, C. and Pironneau, O. Derivatives and control in the presence of shocks. *Computational Fluid Dynamics Journal*, 11, 09 2002.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1994.

Bartholomew-Biggs, M., Brown, S., Christianson, B., and Dixon, L. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124(1):171–190, 2000. ISSN 0377-0427. doi: https://doi.org/10.1016/S0377-0427(00)00422-2. URL https://www.sciencedirect.com/science/article/pii/S0377042700004222. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.*, 18(1):5595–5637, jan 2017. ISSN 1532-4435.

Beck, T. Automatic differentiation of iterative processes. *Journal of Computational and Applied Mathematics*, 50(1):109–118, 1994. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(94)90293-3. URL https://www.sciencedirect.com/science/article/pii/0377042794902933.

Beck, T. and Fischer, H. The if-problem in automatic differentiation. *Journal of Computational and Applied Mathematics*, 50(1):119–131, 1994. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(94)90294-1. URL https://www.sciencedirect.com/science/article/pii/0377042794902941.

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2): 135–151, 2002.

Blonigan, P. J. Adjoint sensitivity analysis of chaotic dynamical systems with non-intrusive least squares shadowing. *Journal of Computational Physics*, 348:803–826, 2017.

Bolte, J. and Pauwels, E. A mathematical model for automatic differentiation in machine learning, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Christianson, B. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3(4):311–326, 1994.

Christianson, B. Reverse aumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

Collis, S., Ghayour, K., Heinkenschloss, M., Ulbrich, M., and Ulbrich, S. *Towards adjoint-based methods for aeroacoustic control*. doi: 10.2514/6.2001-821. URL https://arc.aiaa.org/doi/abs/10.2514/6.2001-821.

Dussault, J.-P. and Hamelin, B. Robust descent in differentiable optimization using automatic finite differences. *Optimization Methods and Software*, 21(5):769–777, 2006. doi: 10.1080/10556780500151984. URL https://doi.org/10.1080/10556780500151984.

Ercegovac, M., Lang, T., Muller, J.-M., and Tisserand, A. Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. *IEEE Transactions on Computers*, 49(7):628–637, 2000. doi: 10.1109/12.863031.

Farrell, P. E., Ham, D. A., Funke, S. W., and Rognes, M. E. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4):C369–C393, 2013. doi: 10.1137/120873558. URL https://doi.org/10.1137/120873558.

Fikl, A., Chenadec, V. L., Sayadi, T., and Schmid, P. A comprehensive study of adjoint-based optimization of non-linear systems with application to Burgers' equation. In *46th AIAA Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, jun 2016. doi: 10.2514/6.2016-3805. URL https://doi.org/10.2514%2F6.2016-3805.

Fischer, H. Special problems in automatic differentiation. In Griewank, A. and Corliss, G. F. (eds.), *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pp. 43–50. SIAM, Philadelphia, PA, 1991. ISBN 0–89871–284–X.

Gal, S. An accurate elementary mathematical library for the IEEE floating point standard. *ACM Transactions on Mathematical Software (TOMS)*, 17(1):26–45, 1991.

Gebremedhin, A. H. and Walther, A. An introduction to algorithmic differentiation. *WIREs Data Mining and Knowledge Discovery*, 10(1):e1334, 2020. doi: https://doi.org/10.1002/widm.1334. URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1334.

Giering, R. and Kaminski, T. Recipes for adjoint code construction. *ACM Trans. Math. Softw.*, 24(4):437–474, dec 1998. ISSN 0098-3500. doi: 10.1145/293686.293695. URL https://doi.org/10.1145/293686.293695.

Gilbert, J. C. Automatic differentiation and iterative processes. *Optimization Methods and Software*, 1(1):13–21, 1992. doi: 10.1080/10556789208805503.

Giles, M. and Ulbrich, S. Convergence of linearized and adjoint approximations for discontinuous solutions of conservation laws. Part 1: Linearized approximations and linearized output functionals. *SIAM Journal on Numerical Analysis*, 48 (3):882–904, 2010. doi: 10.1137/080727464. URL https://doi.org/10.1137/080727464.

Giles, M. B. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pp. 35–44. Springer, 2008.

Griewank, A. A mathematical view of automatic differentiation. *Acta Numerica*, 12:321–398, 2003. doi: 10.1017/S0962492902000132.

Griewank, A. and Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008. ISBN 0898716594, 9780898716597.

Griewank, A., Bischof, C., Corliss, G., Carle, A., and Williamson, K. Derivative convergence for iterative equation solvers. *Optimization Methods and Software*, 2(3-4):321–355, 1993. doi: 10.1080/10556789308805549.

Hager, W. W. Runge–Kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87:247–282, 2000.

Hascoët, L. and Utke, J. Programming language features, usage patterns, and the efficiency of generated adjoint code. *Optimization Methods and Software*, 31:885 – 903, 2016. doi: 10.1080/10556788.2016.1146269.

Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06 (02):107–116, 1998. doi: 10.1142/S0218488598000094. URL https://doi.org/10.1142/S0218488598000094.

Hoffmann, P. H. W. A hitchhiker's guide to automatic differentiation. *Numerical Algorithms*, 72(3):775–811, 2016. doi: 10.1007/s11075-015-0067-6. URL https://doi.org/10.1007/s11075-015-0067-6.

Huot, M., Staton, S., and Vákár, M. Correctness of automatic differentiation via diffeologies and categorical gluing. In Goubault-Larrecq, J. and König, B. (eds.), *Foundations of Software Science and Computation Structures*, pp. 319–338, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45231-5.

Krakos, J. A., Wang, Q., Hall, S. R., and Darmofal, D. L. Sensitivity analysis of limit cycle oscillations. *Journal of Computational Physics*, 231(8):3228–3245, 2012. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2012.01.001. URL https://www.sciencedirect.com/science/article/pii/S0021999112000071.

Krawiec, F., Peyton Jones, S., Krishnaswami, N., Ellis, T., Eisenberg, R. A., and Fitzgibbon, A. Provably correct, asymptotically efficient, higher-order reverse-mode automatic differentiation. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022. doi: 10.1145/3498710. URL https://doi.org/10.1145/3498710.

Lee, W., Yu, H., Rival, X., and Yang, H. On correctness of automatic differentiation for non-differentiable functions. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6719–6730. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4aaa76178f8567e05c8e8295c96171d8-Paper.pdf.

Lee, W., Park, S., and Aiken, A. On the correctness of automatic differentiation for neural networks with machine-representable parameters, 2023.

Lew, A. K., Huot, M., Staton, S., and Mansinghka, V. K. ADEV: Sound automatic differentiation of expected values of probabilistic programs. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023. doi: 10.1145/3571198. URL https://doi.org/10.1145/3571198.

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., and Zemel, R. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pp. 3082–3091. PMLR, 2018.

Lorraine, J., Vicol, P., and Duvenaud, D. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.

Margossian, C. C. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data ¡ining and Knowledge Discovery*, 9(4):e1305, 2019.

Moré, J. J. and Wild, S. M. Do you trust derivatives or differences? *Journal of Computational Physics*, 273:268–277, 2014. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2014.04.056. URL https://www.sciencedirect.com/science/article/pii/S0021999114003325.

Moses, W. and Churavy, V. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. *Advances in Neural Information Processing Systems*, 33:12472–12485, 2020.

Nadarajah, S. and Jameson, A. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. In *38th Aerospace sciences meeting and exhibit*, pp. 667, 2000.

Naumann, U. *The art of differentiating computer programs: an introduction to algorithmic differentiation*, volume 24. SIAM, 2012.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks, 2012. URL https://arxiv.org/abs/1211.5063.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. 2017.

Radul, A., Paszke, A., Frostig, R., Johnson, M. J., and Maclaurin, D. You only linearize once: Tangents transpose to gradients. *Proc. ACM Program. Lang.*, 7(POPL), jan 2023. doi: 10.1145/3571236. URL https://doi.org/10.1145/3571236.

Sweby, P. K. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 21(5):995–1011, 1984.

Taftaf, A., Hascoët, L., and Pascual, V. Implementation and measurements of an efficient fixed point adjoint. 01 2015.

Volder, J. E. The cordic trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, 1959. doi: 10.1109/TEC.1959.5222693.

Wang, F., Zheng, D., Decker, J., Wu, X., Essertel, G. M., and Rompf, T. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *Proc. ACM Program. Lang.*, 3(ICFP), jul 2019. doi: 10.1145/3341700. URL https://doi.org/10.1145/3341700.

Wang, Q. Forward and adjoint sensitivity computation of chaotic dynamical systems. *Journal of Computational Physics*, 235: 1–13, 2013.

Wang, Q., Hu, R., and Blonigan, P. Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations. *Journal of Computational Physics*, 267:210–224, 2014.

Wolfe, P. Checking the calculation of gradients. *ACM Trans. Math. Softw.*, 8(4):337–343, Dec 1982. ISSN 0098-3500. doi: 10.1145/356012.356013. URL https://doi.org/10.1145/356012.356013.