Algorithmic Differentiation Mission Planning (ADMission)

Simon Märtens, Erik Schneidereit, Uwe Naumann

STCE, RWTH Aachen University, Germany

Introduction

Algorithmic differentiation can provide sensitivity analysis, accurate to machine precision for all your favorite simulations. Given there are multiple techniques, like tangent and adjoint modes, check-pointing, preaccumulation, code-gen, etc., it can be challenging to apply AD in an optimal manner. Most of the time there is a more innovative way to combine the techniques, than just running plain Tangent or Adjoint AD.

Algorithmic Differentiation

Let $y = F(x) : \mathbb{R}^n \to \mathbb{R}^m$ be implemented as a differentiable program. Tangent AD yields the Jacobian-free Jacobian-matrix product

$$\mathbb{R}^m \ni \dot{y} = \dot{F}(x, \dot{x}) = F'(x) \cdot \dot{x}$$

whereas Adjoint AD yields the matrix-Jacobian product

$$\mathbb{R}^{1\times n} \ni \bar{x} = \bar{F}(x, \bar{y}) = \bar{y} \cdot F'(x).$$

The Jacobian F' can be accumulated at $\mathcal{O}(F') \leq \mathcal{O}(n) \cdot \textsc{cost}(\dot{F})$ and $\mathcal{O}(F') \leq \mathcal{O}(m) \cdot \textsc{cost}(\bar{F})$ respectively.

Jacobian Chaining

Let's consider F consisting of the elemental functions $F_i : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$, $i = 1, \dots, p$ with the tangents $\dot{F}_i$ and adjoints $\bar{F}_i$. A simple compute graph (DAG) for $p = 2$ could look like this:

$$\begin{tikzcd} x \arrow[r, "F_1"] & t \arrow[r, "F_2"] & y \end{tikzcd}$$

For the easier application of elimination techniques, [3] introduces the *Dual Computational graph* (dual c-graph):

$$\begin{tikzcd} \boxed{x} \arrow[r, dotted] & F_1 \arrow[r] & F_2 \arrow[r, dotted] & \boxed{y} \end{tikzcd}$$

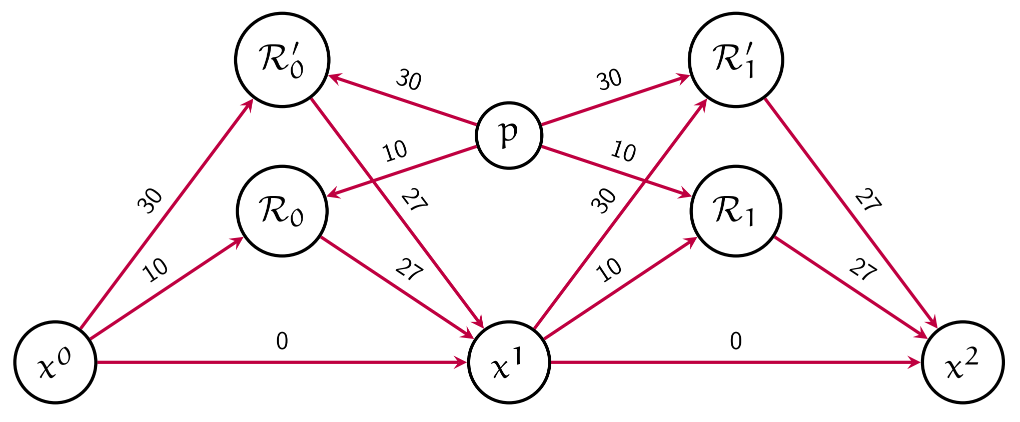
The Jacobian F' can be constructed from the dual c-graph as a Jacobian chain product $F' = F'_2 \cdot F'_1$. Since we have tangents and adjoints, not Jacobians, we need to accumulate F' via the Jacobian-free matrix products that Tangent and Adjoint AD provide. Let's assume $n = n_1 = 4$, $m_1 = n_2 = 2$, $m = m_2 = 32$ and $\textsc{cost}(F_1) = c(F_2) = 100$. Due to the associativity of matrix chain products and the resulting bracketing problem [2] we get the following eight options with their respective costs:

- $F' = \dot{F}_2 \cdot F'_1 = \dot{F}_2 \cdot (\dot{F}_1 \cdot I_{n_1}) \quad \Rightarrow \textsc{cost}(F') = 800$
- $F' = \dot{F}_2 \cdot F'_1 = \dot{F}_2 \cdot (I_{m_1} \cdot \bar{F}_1) \quad \Rightarrow \textsc{cost}(F') = 600$
- $F' = F'_2 \cdot \bar{F}_1 = (I_{m_2} \cdot \bar{F}_2) \cdot \bar{F}_1 \quad \Rightarrow \textsc{cost}(F') = 6400$
- $F' = F'_2 \cdot \bar{F}_1 = (\dot{F}_2 \cdot I_{n_2}) \cdot \bar{F}_1 \quad \Rightarrow \textsc{cost}(F') = 3400$
- $F' = F'_2 \cdot F'_1 = (\dot{F}_2 \cdot I_{n_2}) \cdot (I_{m_1} \cdot \bar{F}_1) \quad \Rightarrow \textsc{cost}(F') = 656$
- $F' = F'_2 \cdot F'_1 = (I_{m_2} \cdot \bar{F}_2) \cdot (I_{m_1} \cdot \bar{F}_1) \quad \Rightarrow \textsc{cost}(F') = 3656$
- $F' = F'_2 \cdot F'_1 = (I_{m_2} \cdot \bar{F}_2) \cdot (\dot{F}_1 \cdot I_{n_1}) \quad \Rightarrow \textsc{cost}(F') = 3856$
- $F' = F'_2 \cdot F'_1 = (\dot{F}_2 \cdot I_{n_2}) \cdot (\dot{F}_1 \cdot I_{n_1}) \quad \Rightarrow \textsc{cost}(F') = 856$

The optimization problem to find the computationally least expensive Jacobian accumulation is NP-complete as proven in [4, 5, 9]. *AD Mission Planning* tries to exploit heuristics to find feasible (persistent memory requirements) and (near-)optimal solutions which prescribe the preferred AD modes (Tangent or Adjoint) for each elemental function.

Result and discussions

For illustration, we consider two Newton steps $x^{i+1} = x^i - \mathcal{R}'(x^i, p)^{-1} \cdot \mathcal{R}(x^i, p)$ applied to a parameterized system of nonlinear equations $\mathcal{R}(x(p), p) = 0$ with twice continuously differentiable residual $\mathcal{R} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and regular Jacobian $\mathcal{R}' \equiv \frac{d\mathcal{R}}{dx}$. With $\mathcal{R}_i = \mathcal{R}(x^i, p)$ the DAG can be depicted as follows:



We chose $m = n = 3$ with estimates for the tangent costs attached to the edges. Adjoints are assumed to be twice as expensive as tangents which tends to be optimistic for real-world applications.

Newton AD mission plan optimization					
Preaccumulate All	Tangent	Adjoint	Greedy	Minimize Fill-in	Branch & Bound
No	1608	1608	2181	1182	1155
Yes (+966)	1404	702	810	486	486

The table lists different optimization configurations. We compare plain adjoint and tangent modes against a Greedy and a Minimize Fill-in heuristic. A branch and bound algorithm gives us the true optimum. While the Greedy heuristic produces even worse results than plain tangents or adjoints, the Minimize Fill-in heuristic is actually very close to the optimum. This is even more impressive when we consider the different run times of the optimizer. The heuristics give us a result in a fraction of a second while the branch and bound algorithm actually took over a day to finish.

We have the option to preaccumulate all elemental Jacobians before we start with the elimination. This will reduce the run time (especially for branch and bound) but result in sub-optimal solutions in almost all cases.

Summary and Outlook

As shown above, without guidance from our ADMission Software one could easily miss out on better performance for their derivative calculations. Even optimizing only the combination of tangent and adjoint modes can improve the performance immensely. A preview of the *ADMission* Software is available on our GitHub page (see QR code). In the future, we'd like to expand the optimization to more AD concepts, like check-pointing, parallel taping, path-wise adjoints, implicit function theorem (see the presentation by Uwe Naumann), etc., and consider higher-order derivatives. Integration into our AD tool dco/c++ [11] is planned as well.

ADMission Pipeline

Our software will eventually provide an entire pipeline that helps the users to optimize their AD code. Four major tasks have to be solved:

1. High-level DAG extraction from arbitrary programs (✗)
2. DAG annotation with run time and memory requirements (✗)
3. AD mission plan optimization (✓)
4. Realization of the optimal AD plan via an AD tool (✗)

The steps should be automated as much as possible but still allow the user to intervene and supply the program with insights about the code.

References

[1] Andreas Griewank and Andrea Walther (2000). "Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition". In: *Frontiers in applied mathematics*

[2] Sadashiva S. Godbole (1973). "On Efficient Computation of Matrix Chain Products". In: *IEEE Transactions on Computers* C-22.9, pp. 864–866. DOI: 10.1109/TC.1973.5009182

[3] Uwe Naumann (Apr. 2004). "Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph". In: *Math. Program.* 99, pp. 399–421. DOI: 10.1007/s10107-003-0456-9

[4] Uwe Naumann (Jan. 2008b). "Optimal Jacobian accumulation is NP-complete". In: *Mathematical Programming* 112, pp. 427–441. DOI: 10.1007/s10107-006-0042-z

[5] Uwe Naumann (2008a). "Call Tree Reversal is NP-Complete". In: *Advances in Automatic Differentiation*. Ed. by Christian H. Bischof et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 13–22. ISBN: 978-3-540-68942-3

[6] Uwe Naumann (2009). "DAG reversal is NP-complete". In: *Journal of Discrete Algorithms* 7.4, pp. 402–410. ISSN: 1570-8667. DOI: https://doi.org/10.1016/j.jda.2008.09.008. URL: https://www.sciencedirect.com/science/article/pii/S1570866708000737

[7] Uwe Naumann (2012). *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. USA: Society for Industrial and Applied Mathematics. ISBN: 161197206X

[8] Uwe Naumann (Jan. 2020). "On Sparse Matrix Chain Products". In: pp. 118–127. ISBN: 978-1-61197-622-9. DOI: 10.1137/1.9781611976229.12

[9] Uwe Naumann (2021). "On the Computational Complexity of the Chain Rule of Differential Calculus". In: *CoRR* abs/2107.05355. arXiv: 2107.05355. URL: https://arxiv.org/abs/2107.05355

[10] Simon Märtens, Erik Schneidereit, and Uwe Naumann (2022). *ADMission*. URL: https://github.com/STCE-at-RWTH/ADMission

[11] Numerical Algorithms Group Ltd (2022). *Derivative Code by Overloading in C++*. URL: https://www.nag.com/content/algorithmic-differentiation-software