Procedia Computer Science

Volume 80, 2016, Pages 2231–2235

ICCS 2016. The International Conference on Computational Science

Algorithmic Differentiation of Numerical Methods: Second-Order Adjoint Solvers for Parameterized Systems of Nonlinear Equations

Niloofar Safiran[1], Johannes Lotz[1], and Uwe Naumann[1]

LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen, Germany
[safiran, lotz, naumann]@stce.rwth-aachen.de

Abstract

Adjoint mode algorithmic (also know as automatic) differentiation (AD) transforms implementations of multivariate vector functions as computer programs into first-order adjoint code. Its reapplication or combinations with tangent mode AD yields higher-order adjoint code. Second derivatives play an important role in nonlinear programming. For example, second-order (Newton-type) nonlinear optimization methods promise faster convergence in the neighborhood of the minimum through taking into account second derivative information. The adjoint mode is of particular interest in large-scale gradient-based nonlinear optimization due to the independence of its computational cost on the number of free variables. Part of the objective function may be given implicitly as the solution of a system of n parameterized nonlinear equations. If the system parameters depend on the free variables of the objective, then second derivatives of the nonlinear system's solution with respect to those parameters are required. The local computational overhead as well as the additional memory requirement for the computation of second-order adjoints of the solution vector with respect to the parameters by AD depends on the number of iterations performed by the nonlinear solver. This dependence can be eliminated by taking a symbolic approach to the differentiation of the nonlinear system.

Keywords: Second-Order Adjoint Nonlinear Solver, Algorithmic Differentiation

1 Introduction and Foundations

In this paper we consider two alternative approaches to evaluating second-order adjoints of numerical simulation programs with an embedded iterative solver for a system of n parameterized nonlinear equations. Naive application of *algorithmic differentiation* (AD) [9, 11] yields a computational cost for second-order adjoints of the solution vector with respect to the parameters which depends on the number of nonlinear iterations ν. *Symbolic differentiation* (SD) is based on the assumption of having reached the exact solution of the nonlinear system. The derived

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.388

formulations allow an implementation which reduces memory requirement and computational cost by an order of complexity.

The focus of this paper is on adjoint mode AD. Analogous results were derived for tangent mode AD in [13]. In the following section we recall some aspects from [12] which this work is based on. Further related work by others includes [3, 4, 6, 8] as well as [9, section 15.5]. We consider the computation of second-order adjoints for solvers of systems of n nonlinear equations depending on m parameters and described by the residual

$$\mathbf{r} = F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) \quad : \quad \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \ . \tag{1}$$

For a given $\boldsymbol{\lambda} \in \mathbb{R}^m$, a *primal solution* vector $\mathbf{x} \in \mathbb{R}^n$ is sought such that $\mathbf{r} = 0$. Without loss of generality, the nonlinear solver is assumed to be embedded into the unconstrained convex nonlinear programming problem (NLP) $\min_{\mathbf{z}} f(\mathbf{z})$, $\mathbf{z} \in \mathbb{R}^q$ for a given objective function $f : \mathbb{R}^q \to \mathbb{R}$. Second-order derivative-based NLP solvers such as Ipopt [14] use the gradient and the Hessian of the objective $y = f(\mathbf{z})$ with respect to the free variables $\mathbf{z} \in \mathbb{R}^q$. Hence, both first and second derivatives of the nonlinear solver are required. As in [12], f is decomposed as $y = f(\mathbf{z}) = p(S(\mathbf{x}^0, P(\mathbf{z})))$, where $P : \mathbb{R}^q \to \mathbb{R}^m$ denotes the part of the computation that precedes the nonlinear solver $S : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $p : \mathbb{R}^n \to \mathbb{R}$ maps the result $\mathbf{x}$ onto the scalar objective y.

We recall some significant elements of AD described in further detail in [9, 11]. Without loss of generality, the following discussion is based on Equation (1). Let $\mathbf{u} = (\mathbf{x}, \boldsymbol{\lambda}) \in \mathbb{R}^h$ and $h = n + m$. AD yields semantical transformations of the given implementation of F as a computer program into first and higher (k-th) derivative code. F is assumed to be k times continuously differentiable. Extensions of AD to nondifferentiable functions have been the subject of ongoing research for several years; see, for example, [7, 10]. AD provides two basic models, the tangent and the adjoint model. Tangents of a variable are denoted by superscripts, e.g. $\mathbf{x}^{(1)}$ for first-order tangents, and adjoints are denoted by subscripts, e.g. $\mathbf{x}_{(1)}$ for first-order adjoints. Higher-order mixed projections carry consequently mixed super- and subscripts, e.g. $\mathbf{x}^{(2)}_{(1)}$ for second-order tangents of adjoints. With the Jacobian $\nabla F = \nabla F(\mathbf{u}) \equiv \frac{\partial F(\mathbf{x},\boldsymbol{\lambda})}{\partial(\mathbf{x},\boldsymbol{\lambda})}$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u})$, $F : \mathbb{R}^h \to \mathbb{R}^n$, the tangent model is given as $\mathbb{R}^h \to \mathbb{R}^n$ defined by $\mathbf{r}^{(1)} \mapsto < \nabla F(\mathbf{u}), \mathbf{u}^{(1)} > \equiv \nabla F(\mathbf{u}) \cdot \mathbf{u}^{(1)}$. The whole Jacobian can be accumulated with machine accuracy at a computational cost of $O(h) \cdot \mathrm{Cost}(F)$ by letting $\mathbf{u}^{(1)}$ range over the Cartesian basis vectors in $\mathbb{R}^h$. The adjoint model is given as $\mathbb{R}^n \to \mathbb{R}^h$ defined by $\mathbf{r}_{(1)} \mapsto < \mathbf{r}_{(1)}, \nabla F(\mathbf{u}) > \equiv \nabla F(\mathbf{u})^T \cdot \mathbf{r}_{(1)}$. As opposed to the tangent model, the adjoint model can compute the whole Jacobian at a computational cost of $O(n) \cdot \mathrm{Cost}(F)$ by letting $\mathbf{r}_{(1)}$ range over the Cartesian basis vectors in $\mathbb{R}^n$. With the Hessian $\nabla^2 F = \nabla^2 F(\mathbf{u}) \equiv \frac{\partial^2 F(\mathbf{x},\boldsymbol{\lambda})}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}$ of $\mathbf{r} = F(\mathbf{u})$ (a symmetric 3-tensor, i.e. $[\nabla^2 F]_{k,i,j} = [\nabla^2 F]_{k,j,i}$) the tangent-over-adjoint model is defined by

$$\mathbf{u}^{(2)}_{(1)} = < \mathbf{r}^{(2)}_{(1)}, \nabla F(\mathbf{u}) > + \sum_{i=1}^{n} \left[\mathbf{r}_{(1)}\right]_i \cdot \nabla^2 [F]_i \cdot \mathbf{u}^{(2)} \ . \tag{2}$$

The whole Hessian can be accumulated with machine accuracy at a computational cost of $O(h \cdot n) \cdot \mathrm{Cost}(F)$ by setting $\mathbf{r}^{(2)}_{(1)} = \mathbf{0}$ and letting $\mathbf{r}_{(1)}$ and $\mathbf{u}^{(2)}$ range over the Cartesian basis vectors in $\mathbb{R}^n$ and $\mathbb{R}^h$, respectively. Re-application of tangent or adjoint mode AD yields higher derivative information. This way, AD can compute projections of derivative tensors of arbitrary order. Our AD tool `dco/c++` is used to implement the theoretical results in this article. `dco/c++` is an AD overloading tool written in and for C++. Apart from arbitrary tangent and adjoint projections, it facilitates the definition of first- and higher-order tangent and adjoint versions

of user-defined high-level intrinsics, for example, second-order adjoint nonlinear solvers. The technical details of the given implementation in dco/c++ are similar to the description in [12] and therefore omitted here.

2 Second-Order Adjoint Nonlinear Solver

We consider two approaches to the generation of second-order adjoint solvers for systems of parameterized nonlinear equations. The first approach is a fully *algorithmic* adjoint version of the solver, that computes second-order adjoints of the approximation of the solution, which is actually computed by the algorithm. In this setting, tangent-over-adjoint AD is applied to the individual statements of the primal nonlinear solver. The memory requirement as well as the computational overhead grows with the number of operations performed by the primal nonlinear solver. The resulting memory requirement is likely to exceed the available resources for most real-world applications. Checkpointing techniques can help keeping the required memory feasible at the expense of additional primal function evaluations [9, 11]. Reverse accumulation [3] limits the memory requirement to that induced by a single iteration of the primal nonlinear solver. Alternatively, one is a coupled approach uses a *symbolic* adjoint version of the solver, based on the assumption that the exact primal solution has been reached. The nonlinear system $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ can be differentiated symbolically in this case according to the *implicit function theorem*. The discrepancies in the numerical results computed by second-order algorithmic and symbolic adjoint nonlinear solvers depend on the accuracy of the approximation of the primal solution. The following theorem shows that second-order adjoints of the paramters can be computed by solving one linear system with the Jacobian and two linear systems with the transposed Jacobian of the residual as system matrix followed by two evaluations of the second-order adjoint residual.

Theorem 2.1. *Let for* $\mathbf{r} = F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) : I\!R^n \times I\!R^m \to I\!R^n$ *and for a given* $\boldsymbol{\lambda} \in I\!R^m$*, a vector* $\mathbf{x} \in R^n$ *be sought such that* $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$*. Second-order adjoints* $\boldsymbol{\lambda}_{(1)}^{(2)} \in I\!R^m$ *of the solution* $\mathbf{x} \in I\!R^n$ *of the system of parameterized nonlinear equations* $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ *with respect to the parameter vector* $\boldsymbol{\lambda} \in I\!R^m$ *are equal to*

$$\boldsymbol{\lambda}_{(1)}^{(2)} =< \mathbf{z}, \frac{\partial^2 F}{\partial \boldsymbol{\lambda} \partial (\mathbf{x}, \boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \mathbf{z}^{(2)}, \frac{\partial F}{\partial \boldsymbol{\lambda}} > , \tag{3}$$

where $\mathbf{z} \in I\!R^n$ *solves* $\left(\frac{\partial F}{\partial \mathbf{x}}\right)^T \cdot \mathbf{z} = -\mathbf{x}_{(1)}$*. Furthermore,* $\mathbf{z}^{(2)} =< \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >\in I\!R^n$ *satisfies* $\left(\frac{\partial F}{\partial \mathbf{x}}\right)^T \cdot \mathbf{z}^{(2)} = - < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x} \partial (\mathbf{x}, \boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > -\mathbf{x}_{(1)}^{(2)}$ *, where* $\mathbf{x}^{(2)} = \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)}$ *is evaluated by a first-order symbolic tangent version of the nonlinear equation [12], i.e.* $\frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(2)} = -\frac{\partial F}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)}$*.*

Proof. First-order adjoints $\boldsymbol{\lambda}_{(1)}$ of the solution $\mathbf{x}$ of $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$ with respect to $\boldsymbol{\lambda}$ are equal to $\boldsymbol{\lambda}_{(1)} =< \mathbf{z}, \frac{\partial F}{\partial \boldsymbol{\lambda}} >= \sum_{i=1}^{n} \mathbf{z}_i \cdot \frac{\partial F_i}{\partial \boldsymbol{\lambda}}^T$, where $\mathbf{z}$ is the solution of the linear system $\frac{\partial F}{\partial \mathbf{x}}^T \cdot \mathbf{z} = -\mathbf{x}_{(1)}$, see [12]. The directional derivative of $\boldsymbol{\lambda}_{(1)}$ with respect to $\boldsymbol{\lambda}$ in direction $\boldsymbol{\lambda}^{(2)}$ becomes

$$\boldsymbol{\lambda}_{(1)}^{(2)} = \sum_{i=1}^{n} \left[\mathbf{z}_i^{(2)} \cdot \frac{\partial F_i}{\partial \boldsymbol{\lambda}}^T + \mathbf{z}_i \cdot \left[\frac{\partial^2 F_i}{\partial \boldsymbol{\lambda} \partial (\mathbf{x}, \boldsymbol{\lambda})} \cdot \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} \right]^T \right] \tag{4}$$

where $\mathbf{x}^{(2)}$ is evaluated according to the Equation given above. Similarly, the directional deriva-

tive of the linear system $\frac{\partial F}{\partial \mathbf{x}}^T \cdot \mathbf{z} = \sum_{i=1}^{n} \mathbf{z}_i \frac{\partial F_i}{\partial \mathbf{x}}^T = -\mathbf{x}_{(1)}$ is given by

$$\sum_{i=1}^{n} \left[\mathbf{z}_i^{(2)} \cdot \frac{\partial F_i}{\partial \mathbf{x}}^T + \mathbf{z}_i \cdot \left[\frac{\partial^2 F_i}{\partial \mathbf{x} \partial (\mathbf{x}, \boldsymbol{\lambda})} \cdot \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} \right]^T \right] = \mathbf{x}_{(1)}^{(2)} \tag{5}$$

yielding the linear system for $\mathbf{z}^{(2)}$. □

The amount of memory required for evaluating the symbolic second-order adjoint $\boldsymbol{\lambda}_{(1)}^{(2)}$ is roughly equal to the amount of memory required by the primal nonlinear solver, i.e, $\sim O(n^2)$ for the Jacobian matrix. In Equation (3), the complexity of evaluating the right hand side is $O(1) \cdot Cost(F)$. The Jacobian needs to be factorized only once for the first-order adjoint and can be reused for evaluating the second-order adjoint. Solving the linear system Equation (2.1) by forward/backward substitution at a cost of $O(n^2)$, the complexity for the overall overhead induced by evaluating the symbolic second-order adjoint $\boldsymbol{\lambda}_{(1)}^{(2)}$ becomes proportional to n^3.

3 Results and Conclusion

Figure 1 shows results based on an one-dimensional time-dependent partial differential equation. Newton's method is used as the nonlinear solver in the implicit timestepping scheme, where in each iteration the Jacobian of the residual is factorized as part of a direct approach to the solution of the embedded linear system. The figure clearly shows the advantage in terms of run time and memory requirements of the symbolic approach.

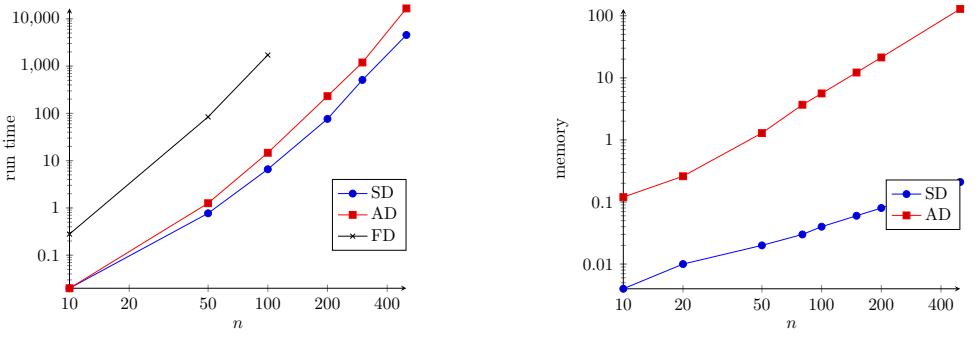

Figure 1: Run time (in seconds) and memory requirements (in MB) for the 1D reference problem using symbolic (SD) and algorithmic (AD) approaches to differentiation as well as finite differences (FD). Missing values indicate failure to converge within 5 hours.

AD is based on the knowledge of partial derivatives of a set of *elemental* functions [9] used to build up an evaluation procedure for a given target function. Traditionally, the built-in functions and arithmetic operators of programming languages have been serving as elementals. Deeper insight into the mathematical structure of numerical simulations yields higher-level elemental functions including linear and nonlinear solvers, optimizers, integrators for ordinary and partial differential equations. First- and higher-order numerical methods generate the need for first and higher derivatives of these new elemental functions. They find application within a wide range of modern methods in Computational Science, Engineering, and Finance ranging from sensitivity and error analysis to adjoint approaches to the solution of inverse

problems. A rich collection of related articles can be found in the proceedings of the last three international conferences on AD [1, 2, 5]. Software tools for AD[1] need to exhibit a high level of flexibility and extensibility in order to facilitate the integration of new elemental functions. Differentiated versions of numerical libraries will soon have to provide appropriate naming and calling conventions for adjoints and adjoints of numerical methods. For example, the Numerical Algorithms Group Ltd.[2] is in the process of expanding the NAG Library with both first- and second-order tangent and adjoint versions for a suitable subset of the more than 1500 numerical routines. `dco/c++` keeps adopting them as new elementals.

Acknowledgments

N. Safiran was financially supported by the Diversity Fund of the RWTH Aachen. J. Lotz was financially supported by the BeProMod project, which is part of the NRW-Strategieprojekt BioSC funded by the Ministry of Innovation, Science and Research of the German State of North Rhine-Westphalia.

References

[1] C. Bischof, M. Bücker, P. Hovland, U. Naumann, and J. Utke, editors. *Advances in Automatic Differentiation*, volume 64 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 2008.

[2] M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50 of *Lecture Notes in Computational Science and Engineering*. Springer, New York, NY, 2005.

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

[4] J. Davies, B. Christianson, L. Dixon, R. Roy, and P. van der Zee. Reverse differentiation and the inverse diffusion problem. *Adv. Eng. Softw.*, 28(4):217–221, 1997.

[5] S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors. *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 2012.

[6] J. Gilbert. Automatic differentiation and iterative processes. *Optimization Methods and Software*, 1:13–21, 1992.

[7] A. Griewank. On stable piecewise linearization and generalized algorithmic differentiation. *Optimization Methods and Software*, 28(6):1139–1178, 2013.

[8] A. Griewank and C. Faure. Reduced functions, gradients and Hessians from fixed-point iterations for state equations. *Numerical Algorithms*, 30(2):113–139, 2002.

[9] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[10] K. Khan and P. Barton. Evaluating an element of the Clarke generalized Jacobian of a composite piecewise differentiable function. *ACM Transactions on Mathematical Software*, 39(4):23–1, 2013.

[11] U. Naumann. *The Art of Differentiating Computer Programs. An Introduction to Algorithmic differentiation*. Number 24 in Software, Environments, and Tools. SIAM, Philadelphia, PA, 2012.

[12] U. Naumann, J. Lotz, K. Leppkes, and M. Towara. Algorithmic differentiation of numerical methods: Tangent and adjoint solvers for parameterized systems of nonlinear equations. *ACM Transactions on Mathematical Software*, 2015. To appear.

[13] N. Safiran, J. Lotz, and U. Naumann. Second-order tangent solvers for systems of parameterized nonlinear equations. *Procedia Computer Science*, 51(0):231 – 238, 2015.

[14] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[1] Refer to the AD community's web portal `www.autodiff.org` for further information.

[2] `www.nag.co.uk`