

Efficient sensitivities for the spin-up phase

Thomas Kaminski¹
Ralf Giering²
Michael Voßbeck³

ABSTRACT In geosciences, it is common to spin up models by integrating with annually repeated boundary conditions. AD-generated code for evaluating sensitivities of the final cyclo-stationary state with respect to model parameters or boundary conditions usually includes a similar iteration for the derivative statements, possibly with a reduced number of iterations. We evaluate an alternative strategy that first evaluates the full Jacobian for the final iteration and from there applies the implicit function theorem to solve for the sensitivities of the cyclo-stationary state. We demonstrate the benefit of the strategy for the spin-up of a simple box-model of the atmospheric transport. We derive a heuristic inequality for this benefit, which increases with the number of iterations and the number of independent variables but decreases with the size of the state space.

Keywords: automatic differentiation, spin-up, sensitivities, source-to-source transformation, TAF, implicit function theorem

In geosciences it is common to spin up models to a cyclo-stationary state. For instance, to simulate the global carbon dioxide distribution in the atmosphere, one runs an atmospheric transport model with a repeated seasonal cycle of carbon dioxide fluxes at the Earth's surface [10]. The spin-up is completed once the simulated seasonal cycle of atmospheric carbon dioxide does no longer change from one year to the next. Other examples are the simulation of the thermo-haline ocean circulation [11], or the slow soil-carbon pool in terrestrial biosphere models [14], which only equilibrate after thousands of years of simulation.

Formally such a model can be represented by a function $f : R^p \times R^n \rightarrow R^n$ mapping the state x at January 1, 0 am, to next year's state at the

¹FastOpt, Hamburg, Germany, (Thomas.Kaminski@FastOpt.com)

²FastOpt, Hamburg, Germany, (Ralf.Giering@FastOpt.com)

³FastOpt, Hamburg, Germany, (Michael.Vossbeck@FastOpt.com)

same time, y :

$$y = f(b, x) \quad , \quad (1.1)$$

where b denotes input quantities other than the initial state such as boundary conditions or parameters in the model.

A necessary condition for terminating the spin-up is that the integration has converged to a fixed point x_f (final state), i.e.

$$x_f = f(b, x_f) \quad (1.2)$$

Let us now assume that we need the sensitivity of the final state x_f with respect to b . This sensitivity might be interesting per se, or it might be a part of a more extended sensitivity computation of other quantities that depend on x_f through a continued integration with x_f as starting point.

Now the most obvious action is to apply an Automatic Differentiation (AD) tool to the entire spin-up code and generate derivative code for evaluating the derivative of x_f with respect to b . Applying our AD-tool Transformation of Algorithms in Fortran (TAF, [5, 6]) in forward mode [7] generates code that iterates

$$dy/db = \partial f/\partial b + \partial f/\partial x * dx/db \quad , \quad (1.3)$$

(with dx/db initialised to 0) along with (1.2), where each relevant statement is preceded by the corresponding derivative statement.

Griewank [7] derives the convergence criteria for (1.3) from the implicit function theorem for (1.2). Moreover he suggests a so-called delayed piggyback strategy, which does not turn on derivative propagation until the iteration of (1.2) converges well. Bücken et al. [2] apply this delayed piggyback strategy to a CFD code.

Christianson [3, 4] analyses reverse mode (adjoint) AD of the iteration of (1.1) and suggests an efficient alternative adjoint, which only uses the required values (trajectory) [5] from the last iteration of (1.1) and thus considerably reduces the resources for storing required variables. TAF implements automatic generation of the Christianson scheme [6].

All above strategies (standard black-box AD, delayed piggyback, Christianson) are matrix-free, i.e. they employ products of the Jacobian $\partial f/\partial x$ with a vector to solve (1.3). The present study explores the alternative strategy of first computing the full Jacobian $\partial f/\partial x$ and then solving (1.3). Like the above mentioned strategies, the full Jacobian strategy is analytically based on the implicit function theorem for (1.2). Presuming that f is sufficiently well behaved and $df/dx(x_f, b)$ is non-zero for a given b , then $\tilde{b} \rightarrow x_f(\tilde{b})$ is well-defined around b , and

$$0 = \partial f/\partial b(x_f(b), b) + (\partial f/\partial x(x_f(b), b) - Id)dx/db \quad , \quad (1.4)$$

where Id denotes the identity in \mathbb{R}^n . The strategy solves (1.4) for dx/db by inverting $(df/dx - Id)$.

TABLE 1.1. Performance for derivatives of final boxmod state with respect to mixing rate

Strategy	Iteration [s]	Derivative [s]	Deriv/Iter	stand./Jacobian
standard	0.4372E+01	0.8324E+01	0.1904E+01	44.0
full Jacobian	0.4372E+01	0.1890E+00	0.4323E-01	1.0

As a test code, we reactivate “boxmod”, a simple model of the atmospheric transport. The model and its forward and reverse mode derivatives are described in [15, 13] and [13], respectively. There is one box for each hemisphere, its concentration takes the role of x in (1.1). The inter-hemispheric mixing rate [13] of 1/year is its single parameter and corresponds to b in (1.1). We use the methyl chloroform setup described in [13], repeating 1978 surface source estimates of Prinn et al. [12] and a uniform sink term corresponding to an inverse lifetime of 1/4.7 years [8].

We carry out performance tests on an Intel Pentium M 1400MHz processor, using the Lahey Fujitsu Fortran 95 compiler lf95 with option `-dbl`, i.e. we run in double precision. To make sure we spend the bulk of CPU-time within boxmod and its derivative code, we do 100000 time steps per year. Each test has been repeated 10 times. Our basic test iterates (1.1) until the relative difference of y and x for both components is less than 10^{-7} . This corresponds to an absolute difference of about 10^{-5} , which is reached after $k = 49$ iterations. To test the standard strategy we generate the derivative of the entire iteration in the TAF forward and pure mode. This means the derivative code does not include a function evaluation, and function code statements are only included where necessary to provide required values.

The second row of Table 1.1 shows the performance for the standard strategy. The second and third column show the CPU times (in seconds) for the entire iteration and its derivative, respectively. The fourth column shows their ratio and the fifth column the ratio of the derivative and the derivative of the third row. The third row in turn lists the equivalent numbers for the full Jacobian strategy, where TAF is also invoked in forward and pure mode to generate code for simultaneous evaluation of the derivatives of f with respect to x and b . Finally dx/db is computed; its value equals the value from the standard strategy to at least 9 digits.

In this example, the full Jacobian strategy outscores the standard strategy by a factor of about 44. Why is that? Let $r(m)$ denote the performance of a Jacobian product with m vectors for a one year run of boxmod. If one used the number of operations as performance measure, $r(\cdot)$ would be an affine function of m , i.e.

$$r(m) = r(1) + s \cdot (m - 1) \quad . \quad (1.5)$$

Switching back to CPU time, our standard measure for performance, the form of $r(m)$ depends on additional factors, most of which are platform and compiler dependent. Examples are data locality, vector length, level of compiler optimisation, and other compiler options. Also, from a certain m the computation will exceed the available memory and hence needs to be split up. Previous performance test for TAF-generated reverse mode Jacobians of a large Fortran-90 code on a different platform yield an $r(m)$ that may indeed be approximated by (1.5) with $s = 1/4$ and $r(1)$ between 3 and 4 (see Figure 4 of [9]).

Our present example is a bit more complex as, in addition to increasing the number of derivatives, we are also increasing the set of quantities with respect to which we are differentiating. On the other hand, the state is active [1, 5] even when differentiating only with respect to b , i.e. derivatives of the state are propagated anyway. Extending the Jacobian from derivatives with respect to b to derivatives with respect to b and x , obviously comes at a low extra cost of a few percent: Neglecting that the ratio of 44 is also influenced by the CPU times spent in the respective solvers, with $k = 49$ iterations we have

$$44 \cdot r(\dim(b) + \dim(x)) \approx 49 \cdot r(\dim(b)) \quad . \quad (1.6)$$

This ratio of 44 depends, of course, crucially on the number of iterations, which in turn depends on the required accuracy for the final state. If one were happy with a relative accuracy of 10^{-3} instead of 10^{-7} , the number of iterations would reduce from 49 to 6 and the ratio from 44 to 5.3.

Whenever (1.5) is a good approximation for capturing the extra cost of the additional derivatives with respect to x , we get the following condition for the full Jacobian strategy to be beneficial:

$$k \cdot (r(1) + s \cdot (p - 1)) > r(1) + s \cdot (p + n - 1) \quad (1.7)$$

$$(k - 1) \cdot (r(1) + p - 1)/s > n \quad . \quad (1.8)$$

For large models, TAF generated forward mode code has a typical $r(1)$ between 1.5 and 3 [6]. The inequality is only relevant for p not much larger than the number of dependents, otherwise one would prefer the reverse mode and derive a corresponding inequality. For only one dependent variable, e.g. a scalar valued objective function of an optimisation problem, we would use the forward mode for $p = 5$ or less. One can maybe hope for a slope s of $1/4$. Given the high number of iterations for spin-up in geosciences and steady aerodynamics problems, which easily reaches a few thousands, the full Jacobian approach looks beneficial for a considerable class of sensitivity calculations.

The idea for this study originates from a discussion with Srikanth Akkaram on sensitivities of a structural mechanics code.

1 REFERENCES

- [1] Christian H. Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3(3):18–32, 1996.
- [2] H. M. Bücker, A. Rasch, E. Slusanschi, and C. H. Bischof. Delayed Propagation of Derivatives in a Two-dimensional Aircraft Design Optimization Problem. In D. Sncchal, editor, *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications and OSCAR Symposium, Sherbrooke, Quebec, Canada, May 11-14*, pages 123–126. NRC Research Press, 2003.
- [3] Bruce Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3:311–326, 1994.
- [4] Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.
- [5] R. Giering and T. Kaminski. Recipes for Adjoint Code Construction. *ACM Trans. Math. Software*, 24(4):437–474, 1998.
- [6] R. Giering, T. Kaminski, R. Todling, and S.-J. Lin. Generating the tangent linear and adjoint models of the DAO finite volume GCM’s dynamical core by means of TAF. *Geophysical Research Abstracts*, 5:11680, 2003.
- [7] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2000.
- [8] S. Houweling, Frank Dentener, and Jos Lelieveld. The impact of non-methane hydrocarbon compounds on tropospheric photochemistry. *J. Geophys. Res.*, 103(D9):10673–10696, 1998.
- [9] T. Kaminski, R. Giering, M. Scholze, P. Rayner, and W. Knorr. An example of an automatic differentiation-based modelling system. In V. Kumar, L. Gavrilova, C. J. K. Tan, and P. L’Ecuyer, editors, *Computational Science – ICCSA 2003, International Conference Montreal, Canada, May 2003, Proceedings, Part II*, volume 2668 of *Lecture Notes in Computer Science*, pages 95–104, Berlin, 2003. Springer.
- [10] R. M. Law, P. J. Rayner, A. S. Denning, D. Erickson, M. Heimann, S. C. Piper, M. Ramonet, S. Taguchi, J. A. Taylor, C. M. Trudinger, and I. G. Watterson. Variations in modelled atmospheric transport of carbon dioxide and the consequences for CO₂ inversions. 10:783–796, 1996.
- [11] M. Nakamura, P. H. Stone, and J. Marotzke. Destabilization of the thermohaline circulation by atmospheric eddy transports. *J. Climate*, 7:1870–1882, 1994.

- [12] Prinn et al. Global average concentration and trend for hydroxyl radical deduction from ALE/GAGE trichloroethane (methyl chloroform) data for 1987–1990. *J. Geophys. Res.*, 97:2445–2461, 1992.
- [13] P. Rayner, R. Giering, T. Kaminski, R. Ménard, R. Todling, and C. Trudinger. Exercises. In P. Kasibhatla et al., editor, *Inverse Methods in Global Biogeochemical Cycles, Geophys. Monogr. Ser.*, volume 114, pages 81–105. American Geophysical Union, Washington, D. C., 2000.
- [14] S. Sitch, I. C. Prentice, B. Smith, A. Arneth, A. Bondeau, W. Cramer, J. O. Kaplan, S. Levis, W. Lucht, M. T. Sykes, K. Thonicke, and S. Venevsky. Evaluation of ecosystem dynamics, plant geography and terrestrial carbon cycling in the LPJ dynamic global vegetation model. *Global Change Biology*, 9:161–185, 2003.
- [15] P. P. Tans. A note on isotopic ratios and the global atmospheric methane budget. *Global Biogeochem. Cycles*, 11:77–81, 1997.