11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
July 20–25, 2014, Barcelona, Spain

ADJOINTS OF FIXED-POINT ITERATIONS

A.Taftaf[*1], **L. Hascoët**[2] **and V.Pascual**[3]
INRIA, Sophia-Antipolis, France, {Elaa.Teftef, Laurent.Hascoet, Valerie.Pascual}@inria.fr

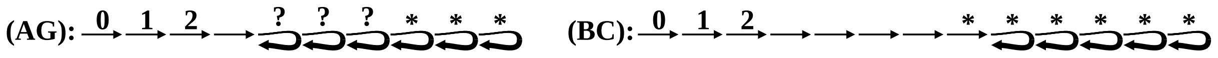
Key words: *Automatic Differentiation, Adjoint, Fixed-Point algorithms*

Adjoint algorithms, and in particular those obtained through the adjoint mode of Automatic Differentiation (AD), are the most efficient way to obtain the gradient of a numerical simulation: assuming that the simulation has a scalar output (objective function), the adjoint algorithm can return its gradient at a cost independent of the number of inputs. The key is that adjoints propagate partial gradients backwards from the result of the simulation. This however uses the data-flow of the simulation in reverse order, at a cost that increases with the length of the simulation. AD research looks for strategies to reduce this cost, taking advantage of the structures of the given program. One such frequent structure is fixed point iterations, which occur at the topmost level of steady-state simulations as well as in unsteady simulations. They may also occur deeper in the simulation, for instance in linear solvers.

It is common wisdom that the first iterations of a fixed-point search operate on a meaningless state vector, and that reversing the corresponding data-flow is wasted effort. An adapted adjoint strategy for the iterative process should consider only the last or the few last iterations. Furthermore, there is a discrete component to an iterative algorithm, namely the number of iterations, and this makes differentiability questionnable. For these reasons we are looking for a specific strategy for the adjoint, that reverses only the necessary data-flow, and that restores confidence in the validity of the derivative.

One could wonder if a differentiation strategy for the iterative computations is really needed in practice, as yet more focused strategies exist. In the case of non-linear solvers using Newton's method, we know only the last iteration need be differentiated. In the case of linear solvers, modern methods such as GMRES hide the iterative process from the differentiation engine: the adjoint need only consist of one or two calls to the same solver. Still, we believe this doesn't cover all cases: steady-state simulations are just one example, where adjoints are much needed. Also, the non-linear variant of GMRES may need a specific adjoint strategy.

At least two authors have studied mathematically fixed point iterations with the goal of defining an efficient adjoint. Griewank's "Delayed Piggyback" (**AG**) [3, ?] ultimately targets computation of the adjoint derivatives together with the tangent derivatives, which are ingredients of a "reduced approximation estimate" that exhibits improved convergence properties. Christianson's "Two Phases"(**BC**) method [1, 2] focuses exclusively on adjoints. Like others, both agree that the iterations convergence rate is similar for the derivative computation and the original computation. Derivatives convergence may lag behind by a few iterations, but will eventually converge at the same rate. The key part is about the scheduling of adjoint iterations with respect to the original ones. Both methods achieve to differentiate *only* the last or the few last iterations i.e. those who operate on physically meaningful values. Both manage also to avoid naïve inversion of the original sequence of iterations, therefore saving the cost of data-flow reversal. Consequently the adjoint, which is itself a fixed point, must have a distinct, specific stopping criterion.



Because of its setting, method **AG** (on the left) makes some additional assumptions whereas **BC** (on the right) remains general on the shape of the iteration step and on the structure of the surrounding program. Another difference is that **BC** starts adjoining the iteration step – actually the last one – only when the original iteration has converged "fully" (*****), whereas **AG** triggers the adjoint iterations earlier, together with the remaining original ones, when those are converged only "sufficiently" (**?**), which may be hard to determine mechanically. Since adjoint computation starts on slightly approximate values, it may require a few more iterations than **BC**. A last difference is that **AG** requires adjoining the sequel of the program i.e. the part after the fixed point iteration, repeatedly inside the adjoint iteration step. This is fine in the chosen setting where the sequel is assumed short, but it has a significant cost in general when the sequel is complex or when fixed point loops are nested.

We are implementing in the team's AD tool TAPENADE a specialized strategy for the adjoint of iterative computations, along the lines of method **BC**, triggered by a user-given differentiation directive. To benchmark and experiment, we selected a steady-state flow solver and a Newton solver.

REFERENCES

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

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

[3] A. Griewank and C. Faure. Piggyback differentiation and optimization. In Biegler et al., editor, *Large-scale PDE-constrained optimization*, pages 148–164. Springer, LNCSE #30, 2003.

[4] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Other Titles in Applied Mathematics, #105. SIAM, 2008.