dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake

Sebastian K. Mitusch[1]**, Simon W. Funke**[1]**, and Jørgen S. Dokken**[1]

1 Simula Research Laboratory

Summary

Adjoint models play an important role in scientific computing. They enable for instance sensitivity and stability analysis, goal-oriented mesh adaptivity and optimisation. However, the derivation and implementation of adjoint models is challenging, especially for models governed by non-linear or time-dependent partial differential equations (PDEs). In (Farrell, Ham, Funke, & Rognes, 2013), the authors proposed to automatically derive adjoint models through high-level algorithmic differentiation, where the forward model is considered as a sequence of variational problems. The implementation, named dolfin-adjoint, automatically and robustly derives adjoint models for models written in the finite element software FEniCS (Logg, Mardal, Wells, & others, 2012). However, the assumption that the model consists of a sequence of variational problems can be limiting. For instance when considering Dirichlet boundary conditions that are not explicitly stated in the variational formulation, when considering complex functionals that cannot be represented as an integral, or when coupling FEniCS to other non-PDE models.

We present a new implementation of dolfin-adjoint that overcomes these limitations. The core of our implementation is a generic, operator-overloading based, algorithmic differentiation tool for Python called pyadjoint. To apply pyadjoint to a Python module, one implements a *pyadjoint.Block* subclass for each module function which can recompute the function with new inputs and compute the function's derivatives. During runtime, pyadjoint builds a graph of Block instances, and applies the chain rule to automatically compute gradients and Hessian actions. Further, pyadjoint includes gradient verification tools and an optimisation framework that interfaces external packages such as scipy, ipopt, moola and ROL.

To support automated adjoints for FEniCS and Firedrake (Rathgeber et al., 2017) models, we overloaded their user-interface functions. In FEniCS and Firedrake, variational problems are represented in the domain-specific language UFL (Alnæs, Logg, Ølgaard, Rognes, & Wells, 2014). UFL allows the definition and manipulation of discrete variational formulations, which we leverage to automatically obtain the desired equations in a format that can be solved by FEniCS/Firedrake. This allows us to efficiently derive the adjoint and tangent-linear equations and solve them using the existing solver methods in FEniCS/Firedrake, as described in (Farrell et al., 2013). In addition, we have implemented support for computing the adjoint solution at the boundary, which enables the automatic differentiation of PDE solutions with respect to strongly imposed Dirichlet boundary conditions.

The dolfin-adjoint repository contains a wide range of tests and demos. The demos are documented and available at www.dolfin-adjoint.org.

Acknowledgements

References

Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., & Wells, G. N. (2014). Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, *40*(2), 9. doi:10.1145/2566630

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

Logg, A., Mardal, K.-A., Wells, G. N., & others. (2012). *Automated solution of differential equations by the finite element method*. Springer. doi:10.1007/978-3-642-23099-8

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T., Bercea, G.-T., et al. (2017). Firedrake: Automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software (TOMS)*, *43*(3), 24. doi:10.1145/2998441