

WEdiff: A Python and C++ package for automatic differentiation

Christopher T. DeGroot¹

¹ Department of Mechanical and Materials Engineering, Western University

DOI: [10.21105/joss.00794](https://doi.org/10.21105/joss.00794)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Submitted: 26 June 2018

Published: 27 June 2018

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC-BY](#)).

Summary

The need to calculate derivatives occurs frequently in the development of all types of codes that do numerical computations. There are a wide variety of methods available to calculate numerical derivatives, including finite differences. Although finite differences can be useful, they are prone to various errors including truncation error (when the difference between the function values is too large) and cancellation error (when the difference between the function values is too small (Griewank and Walther 2008)). Automatic differentiation is a technique that is able to calculate derivatives with machine accuracy by systematically propagating derivatives through the code using the chain rule of calculus. Essentially, the algorithm is viewed simply as a large number of differentiable operations on data, i.e. the composition of a the functions f_0, f_1, \dots, f_n , for which the derivative with respect to the first function value (which may be a constant) is:

$$\frac{\partial f}{\partial f_0} = \frac{\partial f_n}{\partial f_{n-1}} \frac{\partial f_{n-1}}{\partial f_{n-2}} \dots \frac{\partial f_2}{\partial f_1} \frac{\partial f_1}{\partial f_0}$$

The chain rule, expressed in the equation above, can be evaluated in either forward or backwards order. Presently, this software package implements only the forward mode. One method for propagating derivatives is source code transformation, where the base code is parsed and a secondary code is produced (Bischof et al. 1996, Giering and Kaminski (1998), Hascoet and Pascual (2013)). This method, however, is mainly suitable for procedural programming languages. Another method is based on operator overloading where a class is implemented to hold both the computed value and the accumulated derivative, and implements overloaded operators to propagate the derivatives forwards through the code (Jemcov and Mathur 2004, (???), (???)). This method is suitable for object-oriented languages, and just requires that classes and functions be templated to accept the new data type. It is also able to compute derivatives with respect to arbitrary variables, requiring minimal code changes to implement derivative calculations within an existing code.

WEdiff implements forward mode automatic differentiation using a class called `FwdDiff` which stores both the numerical value and accumulated derivative. All standard mathematical operators and functions are implemented for this type. The code is implemented in C++ and can be used in a straightforward manner in the development of other C++ codes. It also includes Python wrappers generated using SWIG (???), enabling it to be used in the development of Python scripts as well.

WEdiff has been designed to be used by researchers in any area of study that involves numerical computations where derivatives are required. As an example, it has recently been used to determine the sensitivity of temperature fields to the thermophysical properties

of solids using a finite-volume-based numerical code (DeGroot 2018). This work can easily be extended to include finite-volume-based solutions of more complex fluid mechanics problems. **WEdiff** could also be used in many other areas, such as solutions of ordinary differential equations, where the derivative of the solution could be obtained with respect to any input parameters. This would enable both optimization and uncertainty analysis to be conducted. The source code for **WEdiff**, along with test cases, examples, and documentation, can be found on BitBucket (???)

Acknowledgements

This work is supported by a research grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants program.

References

- Bischof, C., P. Khademi, A. Mauer, and A. Carle. 1996. “Adifor 2.0: Automatic Differentiation of Fortran 77 Programs.” *IEEE Comput. Sci. Eng.* 3 (3):18–32.
- DeGroot, C.T. 2018. “Automatic Differentiation of a Finite-Volume-Based Transient Heat Conduction Code for Sensitivity Analysis.” *Numer. Heat Transf. Part B Fundam.*
- Giering, R., and T. Kaminski. 1998. “Recipes for Adjoint Code Construction.” *ACM Trans. Math. Softw.* 24 (4):437–74.
- Griewank, A., and A. Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Philadelphia, USA: SIAM.
- Hascoet, L., and V. Pascual. 2013. “The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification.” *ACM Trans. Math. Softw.* 39 (3):20:1–20:43.
- Jemcov, A., and S. Mathur. 2004. “Algorithmic Differentiation of General Purpose CFD Code : Implementation and Verification.” *Eur. Congr. Comput. Methods Appl. Sci. Eng.*, 1–19.