

# Efficient Discretization and Differentiation of Partial Differential Equations through Automatic Functional Differentiation

Kevin Long

Computational Sciences and Mathematics Research Department  
Sandia National Laboratories  
Livermore, CA  
krlong@sandia.gov

April 9, 2004

## Introduction

High-performance algorithms for PDE-constrained optimization [7] require derivatives of the residual with respect to the design variables. These derivatives are not available from most PDE codes, so to obtain derivatives from an existing code one must apply automatic differentiation (AD) to the code. One alternative approach is to write a new code where low-level calculations are templated on a AD-enabled scalar type [3]. Another approach, which we consider here, is represented by the Sundance project [4, 6], in which a PDE is specified in terms of a high-level symbolic language. That symbolic representation can then be differentiated to obtain the gradients required in optimization.

While Sundance has proven successful for many optimization problems (e.g., [1, 4]), it was discovered that its symbolic differentiation scheme suffered from a combinatorial explosion for sufficiently complicated equation sets because it relied on explicit symbolic expansion of derivatives. It was suggested to us by Roscoe Bartlett [2] that “there ought to be a way to use AD to do the symbolic differentiations in Sundance.” In this paper we show how to formulate the discretization of a PDE, or more generally a PDE-constrained inverse problem, such that automatic differentiation can be used in connection with the symbolic problem representation used in Sundance. As will be seen, what is needed for this application is automation of *functional* differentiation, and when a problem includes spatial derivatives we must use a modified form of AD that can handle the interaction between spatial derivatives and functional derivatives. We call this method automatic functional differentiation (AFD). In Sundance we begin with a complete symbolic representation of the problem, so we have at hand a great deal of information that can be used to make AFD calculations very efficient: for instance, we know in advance the Hessian’s sparsity structure at every node in the expression tree.

While it is well-known that PDEs based on a variational principle can be discretized in terms of functional derivatives, our conclusion that the discretization of *any* PDE can be achieved through functional differentiation is perhaps surprising.<sup>1</sup> Therefore, in the next section, we explain the foundations of this method in some detail.

## Obtaining discrete equations through functional differentiation

The weak form of a forward PDE problem will be, in a very abstract form,

$$G(u, v) = \sum_r \int_{\Omega_r} F_r(\{D_\alpha v\}_\alpha, \{D_\beta u\}_\beta, x) = 0 \quad (1)$$

---

<sup>1</sup>This fact was recently rediscovered independently by Kirby and Knepley using a somewhat different approach. [5]

for all  $v$  in some subspace  $V$ . The functions  $F_r$  are homogeneous linear functions of  $v$  and its derivatives, but can be arbitrary nonlinear functions of  $u$  and its derivatives as well as the independent variable  $x$ . We use the notation  $D_\alpha f$  to indicate partial differentiation of  $f$  with respect to the spatial variables indicated by the multiindex  $\alpha$ . When we use a set  $\{D_\alpha u\}_\alpha$  as the argument to  $F_i$  we mean that  $F_i$  may depend on any one or more members of the set of partial spatial derivatives of  $u$ . Finally, the summation is over geometric subregions  $\Omega_r$ . The integrand  $F_r$  may take different functional forms on different subregions; for example it will usually have different functional forms on the boundary and on the interior.

As usual we discretize  $u$  on a finite-dimensional subspace and also consider only a finite-dimensional set of test functions; we then expand  $u$  and  $v$  as a linear combination of basis vectors  $\phi$  and  $\psi$ ,

$$u = \sum_j^N u_j \phi_j(x) \quad (2)$$

$$v = \sum_i^N v_i \psi_i(x). \quad (3)$$

The requirement that (1) holds for all  $v \in V$  is met by ensuring that it holds for each of the basis vectors  $\psi_i$ . Because  $G$  has been defined as a homogeneous function, clearly this condition is met if and only if

$$\frac{\partial G}{\partial v_i} = \sum_r \sum_\alpha \int_{\Omega_r} \frac{\partial F_r}{\partial (D_\alpha v)} D_\alpha \psi_i = 0. \quad (4)$$

Repeating this process for  $i = 1$  to  $N$  gives  $N$  (generally nonlinear) equations in the  $N$  unknowns  $u_j$ . We now linearize (4) with respect to  $u$  about some  $u^{(0)}$  to obtain a system of linear equations for the full Newton step  $\delta u$ ,

$$\frac{\partial G}{\partial v_i} + \left. \frac{\partial^2 G}{\partial v_i \partial u_j} \right|_{u_j^{(0)}} \delta u_j = 0. \quad (5)$$

In the case of a linear PDE (or one that has already been linearized with some alternative linearization such as Oseen), the ‘‘linearization’’ would be done about  $u^{(0)} = 0$ , and  $\delta u$  is then the solution of the PDE.

Writing the above equation out in full, we have

$$\left[ \sum_r \sum_\alpha \int_{\Omega_r} \frac{\partial F_r}{\partial (D_\alpha v)} D_\alpha \psi_i \right] + \sum_j \delta u_j \left[ \sum_r \sum_\alpha \sum_\beta \int_{\Omega_r} \frac{\partial^2 F_r}{\partial (D_\alpha v) \partial (D_\beta v)} D_\alpha \psi_i D_\beta \phi_j \right] = 0, \quad (6)$$

or in a more compact notation

$$\left[ \sum_r \sum_\alpha \int_{\Omega_r} \mathcal{D}_{D_\alpha v} F_r \quad \psi_i \right] + \sum_j \delta u_j \left[ \sum_r \sum_\alpha \sum_\beta \int_{\Omega_r} \mathcal{D}_{D_\alpha v} \mathcal{D}_{D_\beta v} F_r \quad D_\alpha \psi_i D_\beta \phi_j \right] = 0, \quad (7)$$

where  $\mathcal{D}_{D_\beta v} F_r$  indicates the functional derivative of  $F_r$  with respect to the function  $D_\beta v$ . The two bracketed quantities are the load vector  $f_i$  and stiffness matrix  $K_{ij}$ , respectively.

With this approach, we can compute a stiffness matrix and load vector by quadrature provided that we have computed the first and second order functional derivatives of  $F_r$ . Were we free to expand  $F_r$  algebraically, it would be simple to compute these functional derivatives symbolically, and we could then evaluate the resulting symbolic expressions on quadrature points. We have devised an algorithm and associated data structure that will let us compute these functional derivatives using a variant of AD at the symbolic level, saving us the combinatorial explosion of expanding  $F_r$ .

Finally, we note that the method above generalizes immediately to the calculation of gradients required in PDE-constrained optimization.

## Automatic functional differentiation

We have reduced discretization of a PDE, or more generally of a PDE-constrained optimization problem, to the evaluation of functional derivatives such as  $\mathcal{D}_{D_\alpha u} F$ . The main complication in this task is that the arguments of these functional derivatives may not be exhibited explicitly in an unexpanded expression. For example, the expression

$$G(u, D_x u, D_{xx} u) = D_x [u f D_x u] \quad (8)$$

$$= f (D_x u)^2 + u D_x f D_x u + u f D_{xx} u \quad (9)$$

will have nonzero functional derivatives with respect to  $u$ ,  $D_x u$ , and  $D_{xx} u$ , but  $D_{xx} u$  does not appear explicitly in unexpanded form of the expression. Since we have forbidden ourselves from expanding our expressions, we must generate these implicit spatial derivatives on the fly during the process of automatic differentiation.

### Calculation of Spatial Derivatives

The relationship between spatial derivatives and functional derivatives is given by the multivariable chain rule,

$$D_{x_i} g = \mathcal{D}_{x_i} g + \sum_{\alpha} [\mathcal{D}_{D_\alpha u} g] D_{x_i} D_\alpha u. \quad (10)$$

Note the presence of a functional derivative with respect to a coordinate function  $x$ . The operator  $D_x$  applied to an expression  $f$  yields a nonzero result only if  $f$  depends *explicitly* on  $x$ .

The functional derivative with respect to  $D_\beta u$  of a first order spatial derivative  $D_\alpha$ , where  $|\alpha| = 1$ , is obtained by doing functional differentiation on the chain rule (10),

$$\mathcal{D}_{D_\beta u} D_\alpha g = \mathcal{D}_\alpha \mathcal{D}_{D_\beta u} g + \sum_{\gamma} [\mathcal{D}_{D_\gamma u} \mathcal{D}_{D_\beta u} g] D_\alpha D_\gamma u + \delta_{\alpha+\gamma}^\beta \mathcal{D}_{D_\gamma u} g \quad (11)$$

where  $\delta_{\alpha+\gamma}^\beta$  is the Kronecker delta. This process can be extended to higher derivatives.

### Practical Implementation in a Finite-Element Code

With the above rule for functional differentiation of a spatial derivative operator, we are able to regard spatial differentiation as one more ‘‘atomic operation’’ in an expression tree, alongside the usual operations of addition, subtraction, multiplication, division, and function composition. Thus it is possible to evaluate functional derivatives concurrently with evaluation of the expression as in forward-mode AD. Since the calculation of functional derivatives can be done in a single top-down pass through the unexpanded expression tree, the complexity of evaluating functional derivatives will clearly scale linearly with the number of nodes in the expression tree. This linear scaling has been confirmed in our initial experiments with AFD in Sundance.

Several steps may be taken to improve the efficiency of an AFD calculation. First, as noted above, when implemented in a symbolic system such as Sundance where a data structure for the expression tree is formed explicitly, the structure of the tree can be preprocessed to identify the sparsity structure of the Hessian at every node. Second, because in any nontrivial case the calculation of the stiffness matrix and load vector will require quadrature, it is advantageous to batch-process the derivative calculation over quadrature points (and indeed, over many elements at a time) to amortize the overhead of walking the tree.

In principal, this method could be used in any finite-element code structured such that a problem developer provides functions to evaluate the physics-dependent coefficients of combinations of basis functions. We have redesigned Sundance to use AFD for the evaluation of its symbolic expressions and their derivatives, and are in the process of defining an interface allowing other finite-element codes to use Sundance’s symbolic engine to access AFD capability.

## Conclusions and Further Applications

We have developed a novel approach to finite-element discretization in which we obtain stiffness matrices, load vectors, and their gradients through a generalization of AD that can compute implicit spatial derivatives in the course of a calculation. This method gives us the linear algorithmic scalability of AD in a code that uses a symbolic problem description, and facilitates the development of differentiable codes for PDE-constrained optimization.

Finally, we note that while this paper has concentrated on finite-element methods, the AFD algorithm itself, in particular the mixing of spatial and functional derivatives, is independent of the discretization used, and can be applied to any problem in which it is necessary to obtain both spatial and parametric derivatives in an efficient way. In particular, one could imagine writing a finite-difference code in which AFD is used to identify and differentiate the coefficients of each finite-difference stencil. Beyond the field of PDEs entirely, AFD could be used for efficiently evaluating the combinations of spatial and parametric derivatives required in minimizing the energy of molecular configurations given parametrized interaction potentials.

## Acknowledgements

We are grateful to Ross Bartlett for his suggestion that there “ought to be a way” to combine the flexibility and power of Sundance’s symbolic approach with the efficiency of AD, and to Paul Boggs, Ross Bartlett, and Bart van Bloemen Waanders for many fruitful discussions of PDE-constrained optimization.

## References

- [1] V. Akcelik, G. Biros, O. Ghattas, K. Long, and B. van Bloemen Waanders. A variational finite element method for source inversion for convective-diffusive transport. *Finite Elements in Analysis and Design*, 39(8):683–705, 2003.
- [2] Roscoe A. Bartlett. Private communication. 2002.
- [3] Roscoe A. Bartlett. Derivative calculations for an Euler/Roe dissipation scheme. In *US National Conference on Computational Mechanics*, 2003.
- [4] Paul T. Boggs, Kevin R. Long, Stephen B. Margolis, and Patricia A. Howard. Rapid source location for chemical/biological attacks, part 1: The steady-state case. *submitted to SIAM Journal of Optimization*, 2004.
- [5] Robert C. Kirby. Automatic generation of finite element spaces. Seminar at Sandia National Laboratories, 2004.
- [6] Kevin R. Long. Sundance: A rapid prototyping tool for parallel pde-constrained optimization. In L. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders, editors, *Large-Scale, PDE-Constrained Optimization*, volume 30 of *Lecture Notes in Computational Science and Engineering*, Heidelberg, 2003. Springer-Verlag.
- [7] B. van Bloemen Waanders, R. Bartlett, K. Long, P. Boggs, and A. Salinger. Large scale non-linear programming for pde constrained optimization. Technical Report SAND2002-3198, Sandia National Laboratories, 2002.