# From Analytic to Automated Derivatives: a Case Study of the Electrostatic Potential

H. M. Bücker, B. Lang, A. Rasch, and C. H. Bischof

Institute for Scientific Computing
Aachen University of Technology
Seffenter Weg 23
D–52074 Aachen, Germany
Phone: +49 241 80 4913
E-mail: buecker@sc.rwth-aachen.de

## Abstract

Given a large-scale engineering simulation, one is often interested in the derivatives of certain program outputs with respect to certain input parameters, e.g., for sensitivity analysis. From an abstract point of view, the computer program defines a mathematical function whose derivatives are sought. In this note, two cases are investigated where the underlying functions are given by the solution of different two-dimensional electrostatic potential problems. For a rectangular region, the function and its derivatives can be derived analytically. For a function based on an L-shaped region, the function is computed by a simulation code and its derivatives are obtained by automatic differentiation. In general, this powerful technique is applicable if a function is given in the form of a computer program in a high-level programming language such as Fortran, C, or C++. In contrast to numerical differentiation providing approximations based on divided differences, the derivatives computed by automatic differentiation are accurate. Furthermore, automatic differentiation is also shown to be computationally efficient.

## 1 Introduction

Numerical simulation is an essential research tool used, for instance, to understand electromagnetic fields and their applications to technically advanced real-world problems. A key ingredient to a variety of computational techniques is the efficient and accurate evaluation of the derivatives of some given objective function. Examples where gradients, Jacobians, or higher-order derivatives are needed include the solution of nonlinear systems of equations, stiff ordinary differential equations, partial differential equations, and differential-algebraic equations, to name a few.

In large-scale computational electromagnetics, the objective function is typically not available in analytic form but is given by a computer code written in a high-level programming language such as Fortran, C, or C++. Think of the objective function $f$ as the function computed by, say, a (parallel) code consisting of hundreds of thousands lines simulating the electromagnetic field in a complicated three-dimensional geometry. Given such a representation of the objective function, a traditional way to implement an approximation to the exact derivatives is to use divided differences such as

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}, \tag{1}$$

where $h$ is a suitably chosen step size. An advantage of this numerical differentiation approach is its implementation simplicity; roughly speaking, the function $f$ is used as a black box, needed only to be evaluated at some suitably chosen points. However, the accuracy of the approximation depends crucially on the choice of the step size $h$. If $h$ is too large the divided difference approach is only a rough approximation to the exact derivative

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}. \tag{2}$$

On the other hand, if $h$ is too small then $f(x+h)$ is close to $f(x)$ so that cancellation errors occur when using finite-precision arithmetic in the numerator of (1).

If exact derivatives rather than approximations are desired, there are three options: symbolic differentiation, handcoded derivatives, and automatic differentiation. In contrast to numerical differentiation based on divided differences, these three alternatives would yield exact derivative values if arithmetic could be performed in infinite precision on an actual computer.

Symbolic differentiation is usually performed in computer algebra packages such as Macsyma [1], Maple [2], or Mathematica [3]. Symbolic differentiation operates on expressions, taking a formula for $f$ and some of $f$'s input variables, and producing formulae for the partial derivatives of $f$ with respect to these variables. Even after simplification, these formulae typically contain common subexpressions, but redundant re-computation of these subexpressions cannot be avoided as long as one insists on obtaining explicit formulae for the derivatives. If this restriction is lifted, e.g., by letting Maple generating the derivatives in the form of Fortran code, additional variables are introduced. These variables allow the reuse of already computed intermediate results and therefore reduce the computational complexity of the evaluation. But even with these optimizations the applicability of symbolic differentiation is limited to rather simple functions, the reason being the sheer size of the resulting derivative formulae: In most cases there is no possibility to decide beforehand which branch of a conditional statement will be taken for a particular setting of input variables, and therefore the derivative's formula must cover all cases. Thus, nested branches and loops lead to huge expressions for the derivatives, quickly exhausting any available memory. Therefore, computer algebra packages are not capable of generating an expression from a given large-scale computer program.

Another alternative are handcoded derivatives where an analytic expression for the corresponding derivative, $f'$, is deduced first and then implemented in a computer program by hand. Deriving an analytic expression for $f'$ is often inadequate. Moreover, implementing such an analytic expression by hand is challenging, error-prone, and time-consuming. Hence, other approaches are typically preferred.

Automatic differentiation to be described further in this note is a general technique applicable to large-scale computer programs combining accuracy, efficiency, and ease of use. The overall idea of automatic differentiation is to transform a given computer program for a function $f$ into another computer program for the computation of $f$ *and* some desired derivatives $f'$ by adding additional statements.

We consider two electrostatic potential problems to describe the advantages of automatic differentiation. More precisely, assume that the derivatives of the electrostatic potential with respect to the dielectric permeability is sought for two different geometries. A first example whose simple geometry allows writing down the derivatives in analytic form is given as a motivation in Sect. 2. The technique of automatic differentiation is briefly explained in Sect. 3. In Sect. 4, we apply automatic differentiation to the large finite-element code SEPRAN developed at Delft University of Technology to obtain the corresponding derivatives for an L-shaped region where the solution and its derivatives are not available in analytic form.

## 2 Electrostatic Sample Problem

In simple cases the solution of a two-dimensional electrostatic problem can be given in analytic form. Consider a rectangular region with zero charge density consisting of two equally-sized regions, $R_1$ and $R_2$, with homogeneous media of dielectric permeabilities, $\epsilon_1$ and $\epsilon_2$, respectively. The corresponding configuration is depicted in Fig. 1.

Let $\phi_1$ and $\phi_2$ denote the solution of the Laplace equation in $R_1$ and $R_2$, respectively, with boundary conditions

$$\phi_1(y=0)=0, \quad \phi_2(y=2d)=1, \\ \phi_1(y=d)=\phi_2(y=d),$$

and

$$\epsilon_1 \frac{\partial \phi_1}{\partial \mathbf{n}_1} = -\epsilon_2 \frac{\partial \phi_2}{\partial \mathbf{n}_2} \quad \text{at } y=d,$$

where the unit vector $\mathbf{n}_i$ is the outward normal to the region $R_i$. Then it is easily verified that the electrostatic potential $\phi$ depends on only one of the spatial dimensions, $y$, and is given by

$$\phi_1(y) = \frac{\epsilon_2}{\epsilon_1+\epsilon_2} \cdot \frac{y}{d}, \qquad 0 \le y \le d, \tag{3}$$

$$\phi_2(y) = \frac{\epsilon_1}{\epsilon_1+\epsilon_2} \cdot \frac{y}{d} + \frac{\epsilon_2-\epsilon_1}{\epsilon_1+\epsilon_2}, \qquad d \le y \le 2d, \tag{4}$$

where $d$ is the thickness of the two media in $y$-direction. Thus, the derivatives of $\phi$ with respect to $\epsilon_1$

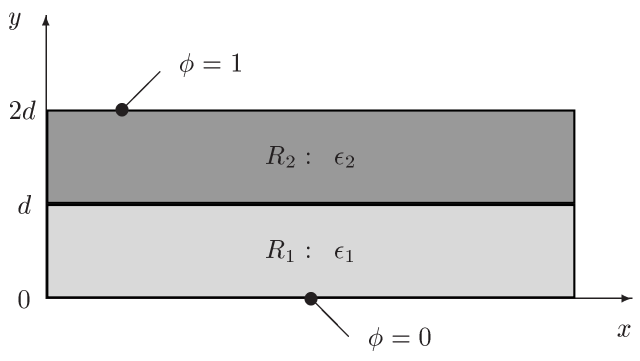

Figure 1: Sample problem $\Delta\phi = 0$.

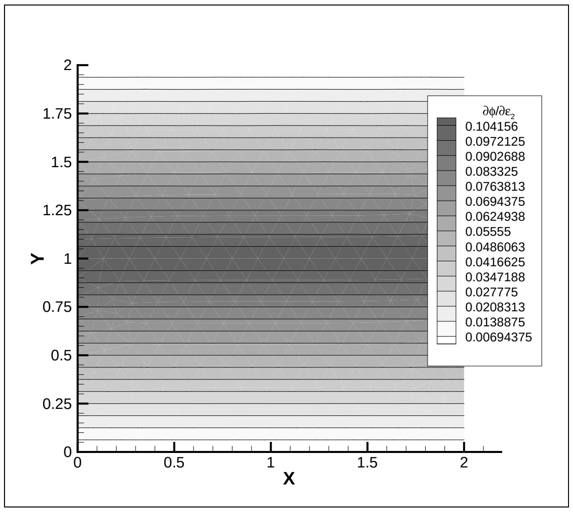

Figure 2: The derivatives $\partial\phi/\partial\epsilon_2$.

and $\epsilon_2$ immediately follow from analytically differentiating the above equations. Differentiating (3) and (4) with respect to $\epsilon_1$ leads to

$$\frac{\partial\phi_1(y)}{\partial\epsilon_1} = \frac{-\epsilon_2}{(\epsilon_1+\epsilon_2)^2}\cdot\frac{y}{d}, \qquad 0 \le y \le d,$$

$$\frac{\partial\phi_2(y)}{\partial\epsilon_1} = \frac{\epsilon_2}{(\epsilon_1+\epsilon_2)^2}\cdot\left(\frac{y}{d}-2\right), \qquad d \le y \le 2d,$$

while differentiating the same equations with respect to $\epsilon_2$ yields

$$\frac{\partial\phi_1(y)}{\partial\epsilon_2} = \frac{\epsilon_1}{(\epsilon_1+\epsilon_2)^2}\cdot\frac{y}{d}, \qquad 0 \le y \le d,$$

$$\frac{\partial\phi_2(y)}{\partial\epsilon_2} = \frac{\epsilon_1}{(\epsilon_1+\epsilon_2)^2}\cdot\left(2-\frac{y}{d}\right), \qquad d \le y \le 2d.$$

Clearly, the derivatives $\partial\phi/\partial\epsilon_1$ and $\partial\phi/\partial\epsilon_2$ depend linearly on $y$. Take the derivatives $\partial\phi/\partial\epsilon_2$, evaluated at

$$\epsilon_1 = 1 \qquad \text{and} \qquad \epsilon_2 = 2, \tag{5}$$

as an example. For a given thickness of the two media, $d = 1$, these derivatives are depicted in Fig. 2. Notice that the largest change of $\phi$ with respect to variations in $\epsilon_2$ appears at the boundary of the two media, $y = 1$.

## 3 Automatic Differentiation

However, if the region is more complicated—such as an L-shaped region—the electrostatic potential $\phi$ is no longer available analytically and it is necessary to compute $\phi$ numerically. Suppose that there is a computer code that, for an L-shaped region, takes the dielectric permeabilities $\epsilon_1$ and $\epsilon_2$ as input and computes the electrostatic potential $\phi$ as output. That is, the function to be differentiated, $\phi$, is implicitly given by a computer program.

Automatic differentiation [4] (AD) is a powerful technique for accurately evaluating derivatives of functions given in the form of a high-level programming language, e.g., Fortran, C, or C++. In automatic differentiation the program is treated as a—potentially very long—sequence of elementary statements such as binary addition or multiplication, for which the derivatives are known. Then, the chain rule of differential calculus is applied over and over again, combining these step-wise derivatives to yield the derivatives of the whole program.

A detailed introduction of AD is beyond the scope of this note and the reader is referred to the book [4] and the workshop proceedings [5, 6, 7]. However, to illustrate the general idea more clearly, consider the following discussion. Suppose that the program for the computation of the electrostatic potential $\phi$ is rewritten as a *straight-line code* which is a finite sequence of elementary operations without loops, conditionals, branching, or subroutines. Straight-line codes may be considered an abstraction of more complicated programs. More precisely, they represent a trace of a particular run of a program for specific values of the input variables. A straight line code has no branches or jumps of any type; that is, every loop is unrolled, every conditional statement is replaced by the appropriate branch, and every subroutine is inlined. Suppose further that complicated statements are broken up into small pieces so that each statement is unary or binary. For instance, the statement

```
z ← x + sin(y)
```

is transformed into the program fragment

```
t ← sin(y)

z ← x + t.
```

Then, for each of these unary or binary statements, an additional statement for the computation of the derivatives is added. In the above example, the program fragment for the code generated by automatic differentiation reads

```
t ← sin(y)

g_t ← cos(y) * g_y

z ← x + t

g_z ← g_x + g_t.
```

Here, new objects `g_t`, `g_y`, `g_z`, and `g_x` containing the derivative values are associated with the objects `t`, `y`, `z`, and `x`, respectively. Properly initializing the derivative objects results in a code capable of computing the original function and its derivatives.

This mechanical process can be automated, and several AD tools are available that augment a given

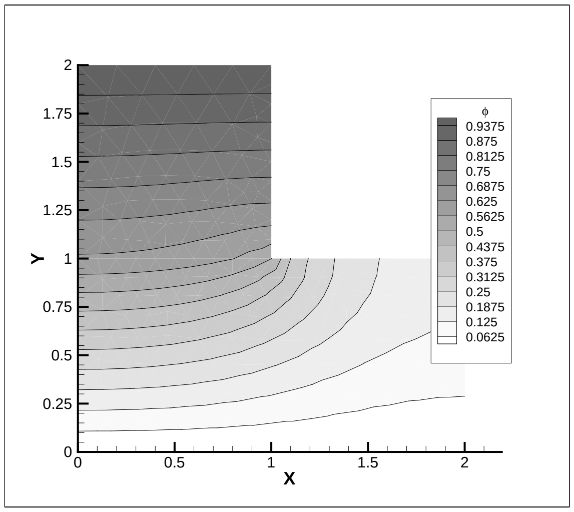

Figure 3: The electrostatic potential $\phi$.

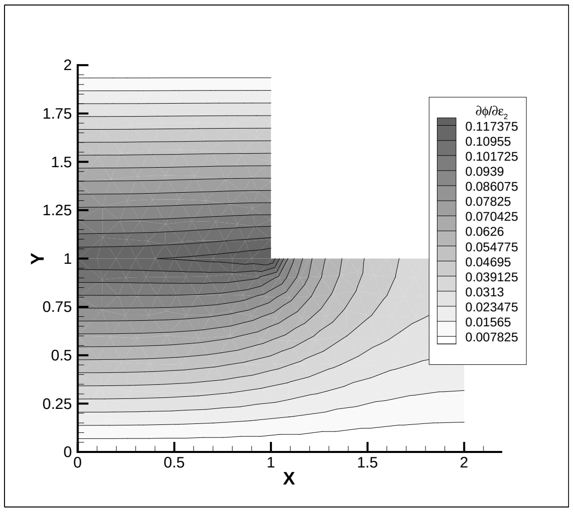

Figure 4: The derivatives $\partial\phi/\partial\epsilon_2$.

code to a new code such that, in addition to the original outputs, the new code also computes the derivatives of some of these output variables with respect to selected inputs. Note that the above example describes the technique from a conceptual point of view and that modern AD tools are much more sophisticated. In particular, the above example demonstrates the basics of the so-called *forward mode* of AD where derivative values are carried forward simultaneously with the values of the function to be differentiated. There is also a so-called *reverse mode* of AD which computes the same numerical results but whose computation time and storage requirement may differ significantly depending on the function at hand. Sophisticated AD tools are also capable of exploiting sparsity involved in Jacobian evaluations. Finally, note that there are similar techniques for higher-order derivatives such as Hessians or the coefficients in the Taylor expansion of a scalar-valued function.

In summary, AD requires little human effort and produces derivatives that are accurate up to machine precision. The reason is that, in AD, there is no concept of a step size $h$ resulting from approximating the exact derivatives (2) by divided differences of type (1). The AD technology is not only applicable for small codes but has been applied successfully to large codes with several hundreds of thousands of lines as described in the following section.

## 4 Differentiating SEPRAN

In the present study, we used the SEPRAN [8] package to compute the electrostatic potential $\phi$ for an L-shaped geometry while still considering a medium consisting of two regions with dielectric permeabilities, $\epsilon_1$ and $\epsilon_2$, with boundary at $y = d$. The results of the simulation using the values for the permeabilities given in (5) are depicted in Fig. 3 where, as before, the thickness is $d = 1$.

SEPRAN is a large general purpose finite element code developed at “Ingenieursbureau SEPRA” and Delft University of Technology and is intended to be used for the numerical solution of second order elliptic and parabolic partial differential equations in two and three dimensions. SEPRAN is used in a wide variety of engineering applications [9, 10, 11, 12, 13, 14, 15] including structural mechanics and laminar or turbulent flow of incompressible liquids. SEPRAN consists of approximately 400,000 lines of Fortran 77.

As in Sect. 2, we are interested in the derivatives of the electrostatic potential with respect to the dielectric permeabilities, $\epsilon_1$ and $\epsilon_2$. However, these derivatives are no longer available analytically because of the more complicated geometry. Rather than employing the traditional technique of divided differencing where one would have to choose a suitable step size, we use automatic differentiation to compute the derivatives. More precisely the ADIFOR [16] system implementing the AD technology was used to transform 400,000 lines of Fortran 77 into a new “differentiated” version of SEPRAN. In Fig. 4, the derivatives $\partial\phi/\partial\epsilon_2$ evaluated at $\epsilon_1 = 1$ and $\epsilon_2 = 2$ are given demonstrating that, again, the largest change of $\phi$ with respect to variations in $\epsilon_2$ appears at the boundary of the two media, $y = 1$. The ratio of the total execution time of the differentiated SEPRAN version and the total execution time of the original SEPRAN code is 2.36. The storage requirement increases by a factor of 2.88 when comparing the original and the differentiated code. We stress that these factors are reasonable keeping in mind that the differentiated SEPRAN code not only computes the original function but also two partial derivatives.

## 5 Concluding Remarks

Automatic differentiation (AD) is a powerful general technique for the computation of derivatives when the underlying function is given in the form of a computer program in a high-level programming language. By transforming a given computer code into a new code capable of computing a function along with its derivatives, AD provides guaranteed accuracy, ease of use, and computational efficiency.

In contrast to numerical differentiation, AD does not involve a divided-difference-like approach thus eliminating the need to experiment with various step sizes. AD is also not to be confused with symbolic differentiation whose applicability is currently limited to rather simple functions.

The traditional approach to adjust model parameters involved in an engineering simulation is to repeatedly run the computer code with various inputs. Automatic differentiation allows the seamless transition from such a mere *simulation* of an engineering system to a systematic adjustment of the model parameters by means of a derivative-based *optimization* approach. That is, automatic differentiation enables, in a completely mechanical fashion, the usage of a Newton-type optimization algorithm where the objective function consists of any given (large-scale) engineering simulation.

In the present study, we applied automatic differentiation to the large general purpose finite element package SEPRAN to obtain a "differentiated" version of SEPRAN. This version was used to compute the derivatives of the electrostatic potential with respect to given dielectric permeabilities. We also point out that, beyond this particular application, the functionality contained in the differentiated version of SEPRAN allows the computation of derivatives of a wide range of potential electrical engineering applications written in SEPRAN in a similarly easy fashion.

## Acknowledgements

The authors would like to thank Jakob Risch for his notable contribution during the initial phase of this project. This research is partially supported by the Deutsche Forschungsgemeinschaft (DFG) within SFB 540 "Model-based experimental analysis of kinetic phenomena in fluid multi-phase reactive systems," Aachen University of Technology, Germany.

## References

[1] R. J. Petti, *Introduction to MACSYMA.* Publishers, Inc., 1997.

[2] M. Kofler, *Maple: An Introduction and Reference.* Addison-Wesley, 1997.

[3] S. Wolfram, *The Mathematica Book.* Cambridge University Press, 4th ed., 1999.

[4] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Philadelphia: SIAM, 2000.

[5] A. Griewank and G. Corliss, *Automatic Differentiation of Algorithms.* Philadelphia: SIAM, 1991.

[6] M. Berz, C. Bischof, G. Corliss, and A. Griewank, *Computational Differentiation: Techniques, Applications, and Tools.* Philadelphia: SIAM, 1996.

[7] G. Corliss, A. Griewank, C. Faure, L. Hascoët, and U. Naumann, eds., *Automatic Differentiation: From Simulation to Optimization.* Springer, 2001. To appear.

[8] G. Segal, *SEPRAN Users Manual.* Ingenieursbureau Sepra, Leidschendam, NL, 1993.

[9] E. G. T. Bosch and C. J. M. Lasance, "High accuracy thermal interface resistance measurement using a transient method," *Electronics Cooling Magazine*, vol. 6, no. 3, 2000.

[10] G. Segal, C. Vuik, and F. Vermolen, "A conserving discretization for the free boundary in a two-dimensional Stefan problem," *Journal of Computational Physics*, vol. 141, no. 1, pp. 1–21, 1998.

[11] A. P. van den Berg, P. E. van Keken, and D. A. Yuen, "The effects of a composite non-Newtonian and Newtonian rheology on mantle convection," *Geophys. J. Int.*, vol. 115, pp. 62–78, 1993.

[12] P. van Keken, D. A. Yuen, and L. Petzold, "DASPK: a new high order and adaptive time-integration technique with applications to mantle convection with strongly temperature- and pressure-dependent rheology," *Geophysical & Astrophysical Fluid Dynamics*, vol. 80, pp. 57–74, 1995.

[13] P. E. van Keken, C. J. Spiers, A. P. van den Berg, and E. J. Muyzert, "The effective viscosity of rocksalt: implementation of steady-state creep laws in numerical models of salt diapirism," *Tectonophysics*, vol. 225, pp. 457–476, 1993.

[14] N. J. Vlaar, P. E. van Keken, and A. P. van den Berg, "Cooling of the Earth in the Archaean: consequences of pressure-release melting in a hot mantle," *Earth Plan. Sci. Lett.*, vol. 121, pp. 1–18, 1994.

[15] C. Vuik, A. Segal, and F. J. Vermolen, "A conserving discretization for a Stefan problem with an interface reaction at the free boundary," *Computing and Visualization in Science*, vol. 3, no. 1/2, pp. 109–114, 2000.

[16] C. Bischof, A. Carle, P. Khademi, and A. Mauer, "ADIFOR 2.0: Automatic differentiation of Fortran 77 programs," *IEEE Computational Science & Engineering*, vol. 3, no. 3, pp. 18–32, 1996.

## 6 Biographies

Prof. Christian Bischof is the director of the Computing and Communication Center of Aachen University of Technology. He also holds a chair for Scientific Computing in the Computer Science Department. Dr. Martin Bücker and Dr. Bruno Lang are researchers, Arno Rasch is a Ph.D. candidate at Prof. Bischof's institute.

Don't hesitate to contact us when you are looking for derivatives of functions given in the form of (large) computer programs.