# Application of Automatic Differentiation to Groundwater Transport Models

Gregory J. Whiffen and Christine A. Shoemaker
*School of Civil and Environmental Engineering, Cornell University, Ithaca, New York*
Christian H. Bischof and Aaron A. Ross
*Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, Illinois*
Alan Carle
*Center for Research on Parallel Computation, Rice University, Houston, Texas*

**Argonne Preprint MCS-P441-0594***

**Abstract.** Automatic differentiation is a technique for generating efficient and reliable derivative codes from computer programs with minimal human effort. Derivatives of model output with respect to input are obtained exactly. No intrinsic limits to program length or complexity exist for this procedure. Calculation of derivatives of complex numerical models is required in system optimization, parameter identification, and systems identification. We report on our experiences with the ADIFOR (Automatic Differentiation of Fortran) tool on a two-dimensional groundwater flow and contaminant transport finite-element model, ISOQUAD, and a three-dimensional contaminant transport finite-element model, TLS3D. Derivative values and computational times for the automatic differentiation procedure are compared with values obtained from the divided differences and handwritten analytic approaches. The automatic differentiation tool ADIFOR produced derivative codes that calculated exact derivatives in typically almost an order of magnitude less CPU time than what is required for the imprecise divided differences method for both the two- and three-dimensional codes. We also comment on the benefit of automatic differentiation technology with respect to accelerating the transfer of general techniques developed for using water resource computer models (such as optimal design, sensitivity analysis, and inverse modeling problems) to field applications.

## 1 Introduction

This paper describes a procedure that requires little human time for generating computationally efficient exact derivatives of a generic water resources simulation model. Calculation of derivatives of computer models is necessary for a wide variety of procedures of interest to numerical modelers. Sensitivity analysis of output parameters with respect to input parameters of complex computer models is often necessary for modelers who perform uncertainty analysis or create stochastic models. Many procedures that optimize performance of an engineered system using a model to predict system performance require derivatives of the same model. Inverse modeling problem methods, in which the goal is to obtain a best set of model input parameters given an observed system behavior, often require derivatives of model output with respect to model input parameters.

Several methods have traditionally been used to obtain derivatives of computer model output with respect to input. Analytical calculation can be used whenever it is reasonable to write analytic derivatives and then code them by hand. *Chang et al.,* [1992], for example, algebraically derived and coded derivatives of a finite element groundwater flow and transport code for use in an optimal control analysis to compute the most cost efficient pumping strategy for groundwater remediation. Often the procedure of deriving and coding analytical derivatives requires substantial human effort and affords great opportunity for error. If a code is extremely complex, this approach may be infeasible.

*Submitted for Publication to Water Resources Research.

Another approach used to obtain derivatives is symbolic differentiation programs (for example, MAPLE and Mathematica.) Symbolic differentiation often generates very large, inefficient derivative formulas. Further, symbolic differentiation may be very difficult to apply to computer models.

Still another approach is hand writing an adjoint code to calculate derivatives. Adjoint codes calculate exact derivatives very efficiently (*Leitmann,* 1981). However, this method also requires a potentially large amount of human effort. Once again, if a model code is very large and complex, this procedure may be infeasible.

Perhaps the most common procedure used to obtain approximate model derivatives of very complex computer models is the divided difference method. The divided difference method calculates derivatives by perturbing model input by small amounts and dividing the resulting model output perturbations by input perturbations (see, for example, *Gorelick et al.,* 1984). This procedure may fail to give accurate derivatives (*Green et al.,* 1993) and is inefficient. The method will require at least as many model simulations as there are input parameters of interest if the Jacobian of interest is dense. However, in general, the divided difference method does not require much human effort and, until recently, has been the only universally applicable approach available to obtain derivatives estimates for very complex computer models.

This paper describes the application of automatic differentiation to obtain codes that efficiently evaluate exact derivatives of complex computer models with a minimum of human effort (*Griewank,* 1989). Automatic differentiation is a method that produces a derivative code given the model code and a list of parameters that are considered dependent and independent variables with respect to differentiation. The method produces a code that will evaluate derivatives exactly (to machine precision). In general, depending on the particular approach chosen, automatic differentiation approaches can compute derivatives with lower arithmetic complexity than that required by the approximate divided differences method. There are no inherent limits on program size or complexity.

It is the purpose of this paper to describe automatic differentiation, to explain its usefulness to water resource problems, and to present numerical results of its application to two groundwater transport codes. The numerical results presented later indicate that automatic differentiation can generate a code that can be used to produce accurate derivatives of the transport codes with a reasonable level of computational efficiency.

### 1.1 Advantages of Automatic Differentiation for Water Resource Problems

General techniques that rely on the output of water resource computer models, such as optimal design, sensitivity or reliability analysis, and inverse modeling problems in ground and surface waters, can all benefit from using automatic differentiation.

Parameter estimation methods such as those developed by *Carrera and Neuman* [1986] for estimation of aquifer parameters involved the coding of an adjoint finite element code to obtain Jacobians. If this and similar procedures are transferred to a new predictive model, a new adjoint finite element code must also be produced. Automatic differentiation can simplify the process of changing the underlying model whenever derivatives are required.

Reliability analysis techniques such as first- and second-order reliability methods applied to water resource models also require sensitivity information or model derivatives. For example, *Jang et al.* [1994] apply these methods to contaminant transport in porous media. Again, automatic differ-

entiation can simplify changing the underlying model. This has enormous potential for accelerating the application of techniques to field problems.

Automatic differentiation can also allow researchers to develop new techniques in less time because little human effort is needed to obtain accurate derivatives. The application of gradient type search optimization procedures (such those of Chang et al., 1992; Gorelick et al, 1984) can benefit. Water flow and transport model sensitivity analysis on new models can be conducted relatively quickly by obtaining derivatives with automatic differentiation. Any researcher who will benefit from exact derivative information from large complex computer codes can potentially benefit from using automatic differentiation.

The following two sections describe the theory behind automatic differentiation and the specific tool ADIFOR (Automatic Differentiation of Fortran) which was tested in the numerical section.

## 2 Automatic Differentiation

Automatic differentiation techniques rely on the fact that every function, no matter how complicated, is executed on a computer as a (potentially very long) sequence of elementary operations such as additions, multiplications, and elementary functions such as *Sine* and *Cosine*. By applying the chain rule,

$$\frac{\partial}{\partial x} f(g(x))|_{x=x_o} = (\frac{\partial}{\partial s} f(s)|_{s=g(x_o)})(\frac{\partial}{\partial x} g(x)|_{x=x_o}), \tag{1}$$

over and over again to the composition of those elementary operations, one can compute derivative information of $f$ exactly and in a completely mechanical fashion.

### 2.1 Simple Example of Automatic Differentiation

The idea behind automatic differentiation is best understood through an example. Assume that we have the sample program shown in Figure 1 for the computation of a function $F : \Re^2 \rightarrow \Re^2$. Here, $x1$ and $x2$ are the independent variables, and $y1$ and $y2$ the dependent variables: $(y1, y2) = F(x1, x2)$. The program in Figure 1 illustrates the cases when automatic differentiation encounters both **if/then** and **do** loop statements. The derivatives of $F$ can be defined everywhere except when $x1 \neq 2$.

If we were to execute this program to compute $F(x1 = 1, x2 = 1.5)$, the list of elementary instructions shown in the column **I** of Table 1 would be executed. Here $r1$ through $r3$ refer to locations where intermediate results are stored. The code shown in Table 1 is a trace (list) of the computations performed to compute $F(1, 1.5)$. As long as $x1 < 2$, this trace can be used as a blueprint for the computation of $F(x_1, x_2)$. To compute derivatives in an automatic fashion, we now associate a unique variable $a_i$, $i = 1, 13$ with each computed value. The variable $a_i$ holds the result of the $i^{th}$ intermediate operation. This value (rounded to three significant digits) is shown in the column **II** labeled $a_i$ in Table 1.

Derivatives can now be computed by associating a value $d_i$ with each intermediate quantity and by using elementary differentiation arithmetic. For example, if we wish to compute $\frac{\partial}{\partial x_1} F(x_1, x_2)|_{(x_1,x_2)=(1,1.5)}$, $d_i$ will hold $\frac{\partial a_i}{\partial x_1}$ for every intermediate quantity $a_i$. Hence, after setting $d_1 = \frac{\partial x_1}{\partial x_1} = 1$ and $d_2 = \frac{\partial x_2}{\partial x_1} = 0$, we can find $d_3$, then $d_4$ and so on until we arrive at $d_{12} = \frac{\partial}{\partial x_1} y1(x_1 = 1, x_2 = 1.5)$

```
10    read x1,x2
20    if ((x1 - 2) > 0) then
30       b = x1
40    else
50       b = 2*x1
60    endif
70       c = 1
80    do i = 1,2
90       c = c + sqrt(c)*b
100   enddo
110   y1 = c/x2
120   y2 = b*x2
```

Figure 1: Sample program to compute $(y1, y2) = F(x1, x2)$

Table 1: Trace of Function Execution

| Code line number | Operation number $i$ | I $i^{th}$ intermediate operation $g_i$ | II Value of operation $a_i$ | III Value of derivative $d_i = \frac{\partial a_i}{\partial x1}$ | IV Value of adjoint $\bar{d}_i = \frac{\partial y1}{\partial a_i}$ |
|---|---|---|---|---|---|
| 10 | 1: | x1 = 1 | 1 | 1 | 4.41 |
| 10 | 2: | x2 = 1.5 | 1.5 | 0 | -2.87 |
| 20 | 3: | r1 = x1 - 2 | -1 | – | – |
| 50 | 4: | b = 2 * x1 | 2 | 2 | 2.21 |
| 80 | 5: | c = 1 | 1 | 0 | 1.05 |
| 90 | 6: | r2 = sqrt(c) | 1 | 0 | 2.10 |
| 90 | 7: | r3 = r2 * b | 2 | 2 | 1.05 |
| 90 | 8: | c = c + r3 | 3 | 2 | 1.05 |
| 90 | 9: | r2 = sqrt(c) | 1.73 | 0.58 | 1.33 |
| 90 | 10: | r3 = r2 * b | 3.46 | 4.61 | 0.67 |
| 90 | 11: | c = c + r3 | 6.46 | 6.62 | 0.67 |
| 110 | 12: | y1 = c / x2 | 4.31 | 4.41 | 1.0 |
| 120 | 13: | y2 = b * x2 | 3 | 3 | 0 |

and $d_{13} = \frac{\partial}{\partial x_1} y2(x_1 = 1, x_2 = 1.5)$ by simple use of the chain rule. For example,

$$\begin{array}{lll} \text{if} & a_j = a_k + a_l, & \text{then} \quad d_j = d_k + d_l. \\ \text{if} & a_j = a_k * a_l, & \text{then} \quad d_j = a_k * d_l + a_l * d_k. \end{array}$$

For univariate functions $g = g(a)$ such as $Sin$, $Cosine$, or $Sqrt$,

$$a_j = g(a_k) \qquad \text{implies} \quad d_j = \frac{\partial}{\partial a_k} g(a_k) * d_k.$$

The values of the $d_j$'s in our particular example are shown in the column **III** labeled $d_i$ in Table 1. After we have traversed all statements, we have computed $\frac{\partial}{\partial x_1} F(x_1 = 1, x_2 = 1.5) = (\frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_1})$, that is, the first column of the Jacobian matrix. From the last two rows of column **III** in Table 1, we see that $(\frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_1})$ is (4.41,3). To obtain the second column of the Jacobian matrix, we initialize $d_1 = \frac{\partial x_1}{\partial x_2} = 0$ and $d_2 = \frac{\partial x_2}{\partial x_2} = 1$ and repeat the previous procedure. Since the propagation of the $d_i$'s is about as costly as that of the $a_i$'s, each Jacobian column calculated costs roughly the same as the evaluation of the original function.

### 2.2 Forward Mode of Automatic Differentiation

This mode of automatic differentiation, where we *maintain the derivatives of intermediate quantities with respect to the independent variables*, is called the *forward mode* of automatic differentiation.

Instead of calculating each column of the Jacobian $J$ separately, we could also have computed all of $J$ in one pass by associating a two-vector storing $\nabla a_j = (\frac{\partial a_j}{\partial x_1}, \frac{\partial a_j}{\partial x_2})^T$ with each intermediate quantity. In general, for a function with $n$ independent variables, we could associate an $n$-vector with each intermediate quantity and then perform a vector operation at each step. This is an advantage when working with a vector processor. If $J$ is dense, the evaluation of $J$ then requires on the order of $n$ times the work that is required to evaluate the function $F$. Often Jacobi matrices are sparse, and sparse storage techniques can be employed rather advantageously. Then the ratio between the cost of evaluating $J$ and $F$ is bounded by the maximum number of nonzeros in any row of the Jacobian. We also mention that if one does not need $J$ per se, but instead $Jv$ for some vector $v$, the fact that differentiation is a linear operator allows us to compute this quantity in one step by initializing $d_i = v_i, i = 1, \ldots, n$.

### 2.3 Reverse Mode of Automatic Differentiation

Another way to compute derivatives is the so-called *reverse mode* of automatic differentiation. Here we *maintain the derivative of the final result with respect to an intermediate quantity.* These quantities are usually called *adjoints*, and they measure the sensitivity of the final result with respect to some intermediate quantity. This approach is closely related to the aforementioned adjoint approach. The discrete analog used in automatic differentiation was apparently first discovered by *Linnainmaa* [1976] in the context of rounding error estimates.

In the reverse mode we associate a scalar $\bar{d}_j$ (say) with each intermediate quantity. We interpret each $\bar{d}_j$ as the derivative of a dependent variable $F$ with respect to the intermediate variable $a_j$

($\bar{d}_j = \frac{\partial F}{\partial a_j}$.) As a consequence of the chain rule it can be shown that for an intermediate quantity $a_j$ whose value is used in the computation of $a_k$, we have

$$\bar{d}_j = \sum_{k \in I} \frac{\partial g_k}{\partial a_j} \bar{d}_k \ , \tag{2}$$

where $g_k$ is the elementary operation that defines $a_k$. The set $I$ is the set $\{k : k > j, \ and \ g_k \ is \ a \ function \ of \ a_j\}$. As an example, assume that we wish to compute $\nabla y_1(x_1 = 1, x_2 = 1.5) = (\frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2})$, that is, the first row of the Jacobian of $F$ (where $F$ is the example given in Figure 1.) We then initialize $\bar{d}_{12} = 1$ and $\bar{d}_{13} = 0$. Since $a_2$ ($= x_2$ in Table 1) is used only in the computation of $a_{12}$ ($= y_1$) and $a_{13}$ ($=y_2$), we can compute from (2)

$$\frac{\partial y_1}{\partial x_2} = \bar{d}_2 = \frac{\partial g_{12}}{\partial a_2} * \bar{d}_{12} + \frac{\partial g_{13}}{\partial a_2} * \bar{d}_{13}.$$

Note terms involving $3 \leq k \leq 11$ in the sum (2) do not appear for this example because these values of $k$ are not elements of the set $I$. Now $g_{12} = a_{11}/a_2$, and $g_{13} = a_4 * a_2$, so

$$\frac{\partial g_{12}}{\partial a_2} = -a_{11}/(a_2)^2$$

and

$$\frac{\partial g_{13}}{\partial a_2} = a_4.$$

By starting from the dependent variables in this fashion, and traversing the computation in reverse order, we emerge at the independent variables with $\nabla y_1$. The adjoint quantities are shown in column **IV** labeled $\bar{d}_i$ in Table 1. The first value under column **IV** is the reverse mode result $\frac{\partial y1}{\partial x1} = 4.41$ and the second value under column **IV** is the reverse mode result for $\frac{\partial y1}{\partial x2} = -2.87$. We can compute $\nabla y_2(x_1 = 1, x_2 = 1.5)$ by repeating this procedure beginning with $\bar{d}_{12} = \frac{\partial y_2}{\partial y_1} = 0$ and $\bar{d}_{13} = \frac{\partial y_2}{\partial y_2} = 1$. Exploiting the sparsity of these vectors, one can bound the ratio between the cost of evaluating $J$ row-wise and that of evaluating $F$ by the maximum number of nonzeros in any column.

We also mention that, in addition to the forward and reverse mode, there are many ways of accumulating derivatives, as a result of the associativity of the chain rule (*Griewank and Reese,* 1991). For example ADIFOR, the tool discussed in the next section uses a combination of both forward and reverse modes.

## 3 The ADIFOR Automatic Differentiation Tool

There have been various implementations of automatic differentiation; an extensive survey can be found in *Juedes* [1991]. In particular, we mention GRESS (*Horwedel,* 1991), and PADRE-2 (*Kubota,* 1991) for Fortran Programs and ADOL-C (*Griewank et al.,* 1990) and ADIC (*Bischof and Mauer,* 1994) for C programs. GRESS, PADRE-2, and ADOL-C implement both the forward and reverse mode. The reverse mode requires one to save or recompute all intermediate values that nonlinearly impact the final result, and to this end these tools generate in some form or another a trace of the computation. The interpretation overhead of this trace and its potentially very large size can be a serious computational bottleneck (*Soulie,* 1991).

Recently, a "source transformation" approach to automatic differentiation has been explored in the ADIFOR (*Bischof et al.,* 1992a; *Bischof et al.,* 1992b), ADIC (*Bischof and Mauer,* 1994) and ODYSSEE (*Rostaing et al.,* 1993) projects. These tools transform code, applying the rules of automatic differentiation, generating new code, which, when executed, computes derivatives without the overhead associated with "tape interpretation" schemes. We employed the ADIFOR tool in our experiments, which, as we will describe shortly, mainly uses the forward mode. In contrast, ODYSSEE employs the reverse mode. The potential "storage explosion" associated with applying the reverse mode to highly nonlinear codes has not been addressed in any tool yet, but the "snapshotting approach" suggested by *Griewank* [1992] has great potential.

ADIFOR provides automatic differentiation for programs written in Fortran 77. Given a Fortran subroutine (or collection of subroutines) describing a "function," and an indication which variables in parameter lists or common blocks correspond to "independent" and "dependent" variables with respect to differentiation, ADIFOR produces Fortran 77 code that allows the computation of the derivatives of the dependent variables with respect to the independent ones. ADIFOR produces portable Fortran 77 code and accepts almost all of Fortran 77, in particular, arbitrary calling sequences, nested subroutines, common blocks, and equivalences. The ADIFOR-generated code tries to preserve vectorization and parallelism in the original code, and employs a consistent subroutine naming scheme which allows for code tuning, the exploitation of domain-specific knowledge, and the use of vendor-supplied libraries. ADIFOR will soon be available via electronic means. Detailed information can be obtained on the world-wide web under URL http://www.mcs.anl.gov/autodiff or by contacting bischof@mcs.anl.gov or carle@cs.rice.edu.

ADIFOR employs a hybrid forward/reverse mode approach to generating derivatives. In this way ADIFOR can achieve some of the computational advantages of both the forward and reverse methods. For each assignment statement, it generates code for computing the partial derivatives of the result with respect to the variables on the right-hand side using the reverse mode approach, and then employs the forward mode to propagate overall derivatives. For example, the single Fortran statement

$$\texttt{y} = \texttt{x(1)} * \texttt{x(2)} * \texttt{x(3)} * \texttt{x(4)} * \texttt{x(5)} \tag{3}$$

embedded in some larger code gets transformed into the code segment shown in Figure 2. ADIFOR uses "$" in naming the new variables required to calculate derivatives. Note that in the reverse mode section of Figure 2, none of the common subexpressions $x(i) * x(j)$ are computed twice even though they are required for the computation of more than one of the derivatives $\frac{\partial\, \texttt{y}}{\partial\, \texttt{x(i)}}$, $i = 1, 2, ..., 5$. This is characteristic of the reverse mode's efficiency when the number of dependent variables is smaller than the number of independent variables. The variable `g$p$` in Figure 2 denotes the number of directional derivatives to be computed (i.e. the number of columns of the desired Jacobian or the desired projection of the Jacobian.) For example, if `g$p$` $= 5$, and the $5 \times 5$ array `g$p` is the $5 \times 5$ identity matrix, i.e. `g$p` $= \frac{dx}{dx}$, then upon execution of these statements, `g$y(i)` equals $\frac{dy}{dx(i)}$. This would be the case if (3) constituted the entire code, and we used ADIFOR to calculate $\frac{dy}{dx(i)}$.

On the other hand, assume that overall we wished only to compute derivatives with respect to a single scalar parameter `s` which appears in statements preceding (3), so `g$p$` $= 1$, and, on entry to

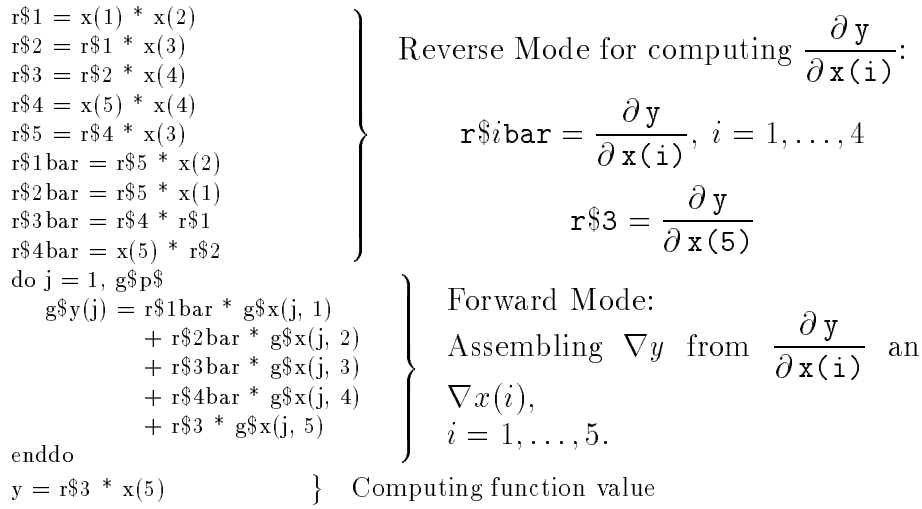


Figure 2: Sample Segment of an ADIFOR-generated derivative code corresponding to the FORTRAN statement given in (3).

this code segment, `g$x(1,i)` $= \frac{dx(i)}{ds}$. Then the do-loop in Figure 2 in effect computes $\frac{dy}{ds}$ as

$$\frac{dy}{ds} = \frac{dy}{dx}\frac{dx}{ds}.$$

The ADIFOR-generated code can be used in various ways (*Bischof and Hovland,* 1991): Instead of simply producing code to compute the Jacobian $J$, ADIFOR can produce code to compute $J * S$, where the "seed matrix" $S$ is initialized by the user. If the user desires the full Jacobian then $S$ can be input as the identity and the ADIFOR-generated code will compute the full Jacobian. If the user only requires the value of the Jacobian times a given vector (as is necessary for some Newton procedures for example) then $S$ can be input as just a vector and the ADIFOR-generated code will compute the product of the Jacobian by that vector. "Compressed" versions of sparse Jacobians can be computed by exploiting the same graph coloring techniques that are used for divided difference approximations of sparse Jacobians (*Averick et al.,* 1994) by carefully selecting the seed matrix $S$. The running time and storage requirements of the ADIFOR-generated code are roughly proportional to the numbers of columns of $S$, which equals the `g$p$` variable in the sample code in Figure 2. Hence the computation of Jacobian-vector products and compressed Jacobians requires much less time and storage than the generation of the full Jacobian matrix.

Experiences with ADIFOR on aeronautics fluid and structures codes have been reported in *Barthelemy and Hall,* [1992]; *Bischof et al.,* [1992c]; *Bischof et al.,* [1994]; *Carle et al.,* [1994]; and *Green et al.,* [1993]. In *Bischof et al.,* [1994] it is also shown how one can use the flexibility of the ADIFOR interface to exploit parallelism to decrease turnaround time for Jacobian computations.

## 4 Differentiated Models: Two- and Three-dimensional Finite Element Models

Finite element models were chosen to test ADIFOR due to the inherent complexity of finite element codes and the corresponding difficulty of obtaining exact derivative information. Two finite element models were chosen to differentiate. We tested the accuracy and computational speed of the code produced by ADIFOR. One two- and one three-dimensional model were selected. Several different mesh sizes were used to test the scalability of the derivative code produced by ADIFOR.

### 4.1 ISOQUAD: 2D Groundwater Flow and Transport Models

ISOQUAD is a two-dimensional (vertical dimension averaged) Galerkin finite-element model of groundwater transient flow and transport (see *Pinder and Frind* [1972] and *Pinder and Gray* [1977].) The flow equation assumed by the model represents a two-dimensional confined aquifer with storage, which, employing the nomenclature of Table 2, can be expressed as

$$\nabla \cdot (bK\nabla \overline{h}) + \sum_{i \in Q} \overline{q_i}\delta(x_i, y_i) - bS_s \frac{\partial \overline{h}}{\partial t} = 0. \tag{4}$$

The governing equation for transport used by the model represents conservative transport with a choice of adsorbtion isotherms. We used a linear isotherm:

$$\nabla \cdot (\theta b \overline{D} \cdot \nabla c) - bv \cdot \nabla c - b(\theta + \rho_B \frac{\partial S}{\partial c}) \frac{\partial c}{\partial t} = 0\,. \tag{5}$$

Table 2: Flow and Transport Model Variables

| | |
|---|---|
| $b$ | saturated vertical thickness of the aquifer |
| $c$ | contaminant concentration |
| $\overline{D}$ | hydrodynamic dispersion tensor |
| $\delta(x_i, y_i)$ | two-dimensional Dirac delta function centered at $x_i, y_i$ |
| $\overline{h}$ | vertically averaged hydraulic head |
| $K$ | hydraulic conductivity tensor |
| $\overline{q_i}$ | pumping rate for well $i$ located at $(x_i, y_i)$ |
| $\rho_B$ | bulk density of the porous media |
| $S$ | mass of adsorbed pollutant per unit mass of porous medium |
| $S_s$ | specific storage |
| $\theta$ | porosity |
| $v$ | Darcy velocity defined as $v = -K \cdot \nabla \overline{h}$. |

ISOQUAD is written in Fortran 77 and is on the order of two thousand lines of code. The model assumes the aquifer is confined but can allow for leakage. The model has been used extensively in the optimal design research (*Chang et al.* [1992], *Culver and Shoemaker* [1992], and *Whiffen and*

*Shoemaker* [1993].) Analytic expressions that can be coded for the derivatives of ISOQUAD are provided by *Chang et al.* [1992]. These hand-coded, validated derivatives have been optimized for performance and allow a comparison and validation of ADIFOR generated derivatives and divided differences derivatives.

The model ISOQUAD is used in its implicit time-stepping mode. In general, implicit time stepping results in difficult-to-derive, dense Jacobians. Explicit time stepping will, in general, result in easier-to-derive, sparse Jacobians. The model has as input the hydraulic heads and single species concentrations at $n$ active nodes and the pumping rates of wells located on $m$ computational nodes. Outputs include contaminant concentrations and hydraulic head values at each of $n$ active nodes for an advanced time step of the simulation. Automatic differentiation was used to obtain the following derivatives:

$$\frac{\partial h_{t+1}}{\partial h_t} \quad \epsilon \ \Re^{n \times n} \tag{6}$$

$$\frac{\partial c_{t+1}}{\partial h_t} \quad \epsilon \ \Re^{n \times n} \tag{7}$$

$$\frac{\partial c_{t+1}}{\partial c_t} \quad \epsilon \ \Re^{n \times n} \tag{8}$$

$$\frac{\partial c_{t+1}}{\partial q_t} \quad \epsilon \ \Re^{n \times m} \tag{9}$$

$$\frac{\partial h_{t+1}}{\partial q_t} \quad \epsilon \ \Re^{n \times m}. \tag{10}$$

The vectors $c_t$ and $c_{t+1} \in \Re^n$ are the values of the contaminant concentration at model time steps $t$ and $t+1$. The vectors $h_t$ and $h_{t+1} \in \Re^n$ are the hydraulic heads at times $t$ and $t+1$. The vector $q_t \in \Re^m$ is the pumping rate at each designated well node at time step $t$. The evaluation of these derivatives is necessary for the optimal control analysis used by *Chang et al.* [1992]. The evaluation of all five derivatives, (6)—(10), together is considered one derivative evaluation in the numerical results that follow.

We choose a mesh of 60 elements and 77 nodes to test automatic differentiation (see Figure 3). In this example, $n = 63$, the number of active nodes. We choose $m = 18$ nodes to locate pumping wells. For this example there were 144 independent variables and 126 dependent variables.

As shown in Figure 4, automatic differentiation provided a code that calculated model derivatives exactly in much less time than the imprecise divided differences method. The ADIFOR-generated code produced derivatives that were the same as the validated handwritten code to the order of the machine precision. In particular, for the 77-node mesh, the ADIFOR code calculated derivatives (6)—(10) in about the time it would take to run the original simulation model 17 times. The same derivatives using the one-sided divided-differences approach require 144 simulations, one for each independent variable. All codes were run on an IBM ES-9000 Mainframe. Figure 4 presents a comparison of CPU times for the ISOQUAD example. If the more accurate centered divided differences approach is used, 288 simulations are necessary. The automatic differentiation code was somewhat slower than the optimized handwritten code by *Chang et al.* [1992], which requires about the same CPU time as 5 simulations.

### 4.2 TLS3D: 3D Advection Diffusion Model

TLS3D is a model of the three-dimensional advection diffusion equation. TLS3D employs a Taylor least-squares finite element procedure to solve the unsteady advection diffusion equation

$$\frac{dc}{dt} = \nabla v \cdot \nabla c + \nabla \cdot D \nabla c. \tag{11}$$

The model uses a three-dimensional serendipity Hermite element for an eight-node hexahedron and was developed to produce accurate results for advection-dominated problems. For a complete description of the Taylor least-squares finite element procedure and three-dimensional model, see *Park and Liggett* [1991]. The code provided to the authors was written in Fortran 77 and is approximately 2300 lines in length.

The model was used on three-dimensional meshes ranging in size from 3 to 128 rectangular box elements to test the scalability of the of the code generated by ADIFOR. The model has as input the three-dimensional flow field velocity vectors $v_t$ for each simulation time step $t$, and initial contaminant concentration at each of $n$ nodes, $c_t$. Model output is single-species contaminant concentrations at each active node, $c_{t+1}$. Automatic differentiation was used to evaluate the following derivative:

$$\frac{\partial c_{t+1}}{\partial v_t} \quad \epsilon \ \Re^{n \times 3n}. \tag{12}$$

Automatic differentiation provided a code that can calculate model derivatives exactly in much less time than the imprecise divided differences. Five different-sized discretizations consisting of 3, 7, 32, 72 and 128 three-dimensional elements were used to compare derivative performances. Figure 5 presents a perspective of the 128-element mesh. The number of independent variables (velocity components) equals three times the number of nodes in the model. The number of independent variables with respect to differentiation for the 3, 7, 32, 72, and 128 element cases are 48, 96, 243, 432, and 675, respectively.

Table 3 and Figure 6 show the results on a SPARCstation 10 model 30 and a single node of the IBM SP1 parallel computer. The columns of Table 3 labeled "1 Simulation" show the run time (in seconds) of the simulation, the columns labeled $\frac{ADIFOR}{DivDiff}$ show the average time required for the ADIFOR code derivatives with respect to one independent variable divided by the time necessary for one simulation. This ratio is equivalent to the ratio of the CPU time required by the ADIFOR code to the CPU time required by the one sided divided differences procedure. These results were obtained when the ADIFOR-generated code is used to generate derivatives with respect to 48 independent variables at a time using an appropriate seed matrix $S$ and then dividing the CPU time by 48 times the CPU time for a single simulation.

We see that the ADIFOR-generated code is more than 5 (7) times faster than the one-sided divided difference approximations on the SPARCstation 10 (SP1 node). The ratio between the ADIFOR-generated derivative code and one-sided divided differences generally decreases with increasing mesh size, from .21 for the 7-element case to .13 for the 128-element case on a single node of a SP1. No hand-coded analytic derivatives of TLS3D are available for comparison. ADIFOR derivative values were nearly exactly the same as the divided differences derivative values using small perturbations.

Table 3: Performance results for TLS3D, simulation times are given in seconds

| Number of Elements (Number of independents) | SPARCstation 10 | | SP/1 Node | |
|---|---|---|---|---|
| | 1 Simulation | $\frac{ADIFOR}{DivDiff}$ | 1 Simulation | $\frac{ADIFOR}{DivDiff}$ |
| 3 (48) | 2.0 | 0.20 | 0.27 | 0.17 |
| 7 (96) | 4.5 | 0.21 | 0.51 | 0.21 |
| 32 (243) | 30.9 | 0.20 | 4.72 | 0.15 |
| 72 (432) | 77.9 | 0.19 | 13.3 | 0.14 |
| 128 (675) | 140.8 | 0.19 | 22.8 | 0.13 |

## 5 Conclusions

We found that automatic differentiation can provide accurate and efficient derivative codes for the complex finite element codes ISOQUAD and TLS3D. In particular, we found ADIFOR generated codes that calculated derivatives in approximately 13% of the time required by divided differences for the two-dimensional model ISOQUAD, and between 13% and 21% of the time required by divided differences for the three-dimensional model, TLS3D. As a result of our investigations, we believe that automatic differentiation can greatly reduce the computational and human time necessary to obtain exact sensitivity informatioe for complex models. Further, automatic differentiation can facilitate model changes by providing a mechanism for generating accurate and efficient derivative codes for new models with very little human effort. Automatic differentiation technology can greatly accelerate the transfer of general techniques developed for using water resource computer models (such as optimal design, sensitivity analysis, and inverse modeling problems) to field problems. Automatic differentiation can also accelerate the rate at which algorithm and model development and testing can occur by providing exact sensitivity information that may not otherwise be available for complex models.

*Acknowledgments.* Primary support for G. J. W. was in the form of a fellowship awarded by the Department of Energy Computational Science Graduate Fellowship Program. This research was funded in part by a grant to C. A. S. from NSF (ASC 8915326) and from the IBM Environmental Research Program. C. H. B.'s work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Aerospace Agency under Purchase Order L25935D. A. C. was supported by the National Aerospace Agency under Cooperative Agreement No. NCCW-0027, and the National Science Foundation through the Center for Research on Parallel Computation by NSF Cooperative Agreement No. CCR-9120008. A. A. R. was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and the National Science Foundation through the Center for Research on Parallel Computation by NSF Cooperative Agreement No. CCR-9120008. The authors gratefully acknowledge the use of the Argonne High-Performance Computing Research Facility. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

TLS3D code was provided by Dr. Nam-Sic Park and Dr. James A. Liggett. The groundwater remediation optimal control model was modified from Fortran code provided by L. Chang. Computer time was provided by the Cornell National Supercomputer Facility (CNSF) through a Strategic User Award to Professor C. A. Shoemaker. The CNSF is funded by the National Science Foundation, International Business Machines, New York State, and The Corporate Institute. We thank Liang-Cheng Chang, Dr. Nam-Sic Park, and Jeanine Plummer for their assistance during this project.

## References

[1] Averick B., J. Moré, C. Bischof, A. Carle, and A. Griewank. Computing large sparse Jacobian matrices using automatic differentiation, *SIAM Journal of Scientific Computing,* 15(2):285–294, 1994.

[2] Barthelemy, J.-F. and L. Hall. Automatic differentiation as a tool in engineering design. In *Proceedings of the 4th AIAA/USAF/NASA/OAI Symp. on Multidisciplinary Analysis and Optimization, AIAA 92-4743.* American Institute of Aeronautics and Astronautics, 1992.

[3] Bischof, C., A. Carle, G. Corliss, and A. Griewank. ADIFOR: Automatic differentiation in a source translator environment. In Paul Wang, editor, *International Symposium on Symbolic and Algebraic Computing 92*, pages 294–302, Washington, D.C., 1992a. ACM.

[4] Bischof, C. , A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR: Generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):11–29, 1992b.

[5] Bischof, C., G. Corliss, L. Green, A. Griewank, K. Haigler, and P. Newman. Automatic differentiation of advanced CFD codes for multidisciplinary design. *Journal on Computing Systems in Engineering*, 3(6):625–638, 1992c.

[6] Bischof, C., L. Green, K. Haigler, and T. Knauff. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. Preprint MCS-P419-0294, Mathematics and Computer Science Division, Argonne National Laboratory, February, 1994. To appear in the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization.

[7] Bischof, C. and P. Hovland. Using ADIFOR to compute dense and sparse Jacobians. ADIFOR Working Note #2, MCS–TM–158, Mathematics and Computer Science Division, Argonne National Laboratory, 1991.

[8] Bischof, C. and A. Mauer, Unpublished information, Argonne National Laboratory, 1994.

[9] Carle, A., L. Green, C. Bischof, and P. Newman, Applications of automatic differentiation in CFD, Proceedings of the 25th AIAA Fluid Dynamics Conference, AIAA Paper 94-2197, 1994.

[10] Carrera, J. and S. P. Neuman, Estimation of Aquifer Parameters Under Transient and Steady State Conditions: 2. Uniqueness, Stability, and Solution Algorithms, *Water Resour. Res.,* 22(2):211-227, 1986.

[11] Chang, L.-C., C. A. Shoemaker, and P. L.-F. Liu, Application of a constrained optimal control algorithm to groundwater remediation, *Water Resour. Res.,* 28(12):3157–3173, 1992.

[12] Gorelick, S. M., C. I. Voss, P. E. Grill, W. Murray, M. A. Saunders, and M. H. Wright, Aquifer reclamation design: The use of contaminant transport simulation combined with nonlinear programming, *Water Resour. Res.,* 20(4):415–427, 1984.

[13] Green, L., P. Newman, and K. Haigler. Sensitivity derivatives for advanced CFD algorithm and viscous modeling parameters via automatic differentiation. In *Proceedings of the 11th AIAA Computational Fluid Dynamics Conference, AIAA Paper 93-3321.* American Institute of Aeronautics and Astronautics, 1993.

[14] Griewank, A. On automatic differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108, Amsterdam, 1989. Kluwer Academic Publishers.

[15] Griewank, A. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods & Software*, 1(1):35–54, 1992.

[16] Griewank, A., D. Juedes, and J. Srinivasan. ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. Preprint MCS-P180-1190, Mathematics and Computer Science Division, Argonne National Laboratory, 1990.

[17] Griewank, A. and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, Penn., 1991.

[18] Horwedel, J. E. GRESS: A preprocessor for sensitivity studies on Fortran programs. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 243–250. SIAM, Philadelphia, Penn., 1991.

[19] Jang, Y-S, N. Sitar, and A. Der Kiureghian, Reliability Analysis of Contaminant Transport in Saturated Porous Media, *Water Resour. Res.,* 30(8):2435-2448, 1994.

[20] Juedes, D. A taxonomy of automatic differentiation tools. In Andreas Griewank and George Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 315–330, SIAM, Philadelphia, Penn., 1991.

[21] Kubota, K. PADRE2, a FORTRAN precompiler yielding error estimates and second derivatives. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 251–262. SIAM, Philadelphia, Penn., 1991.

[22] Leitmann, G., The Calculus of Variations and Optimal Control, Vol. 20 of **Mathematical Concepts and Methods in Science and Engineering**, Plenum Press, New York, 1981.

[23] Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT (Nordisk Tidskrift for Informationsbehandling)*, 16(1):146–160, 1976.

[24] Park, N. -S. and J. A. Liggett, Application of Taylor-least squares finite element to three-dimensional advection-diffusion equation, *Int. J. Numer. Methods Fluids,* 13:759–773, 1991

[25] Pinder, G. F. and Frind, Application of Galerkin's procedure to aquifer analysis, *Water Resour. Res.,* 8(1):108–120, 1972.

[26] Pinder, G. F. and W. G. Gray, *Finite Element Simulation in Surface and Subsurface Hydrology,* Academic Press, Orlando, 1977.

[27] Rostaing, N., S. Dalmas, and A. Galligo. Automatic differentiation in ODYSEE. *Tellus*, 45a(5):558–568, October 1993.

[28] Soulie, E. User's experience with Fortran compilers for least squares problems. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 297–306. SIAM, Philadelphia, Penn., 1991.

[29] Unger, E. R. and L. E. Hall. The use of automatic differentiation in an aircraft design problem, 1994. Extended abstract submitted for the 5th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization.

[30] Whiffen, G. J., and C. A. Shoemaker, Nonlinear Weighted Feedback Control of Groundwater Remediation Under Uncertainty, *Water Resour. Res., 29*(9) 3277-3289, 1993.