PROCEEDINGS, TOUGH Symposium 2003
Lawrence Berkeley National Laboratory, Berkeley, California, May 12–14, 2003

APPLICATION OF AUTOMATIC DIFFERENTIATION IN TOUGH2

Jong G. Kim[1] and Stefan Finsterle[2]

[1]Argonne National Laboratory
9700 S. Cass Avenue, Mail Stop Bldg 362/E325
Argonne, IL 60439
e-mail: jgkim@anl.gov

[2]Lawrence Berkeley National Laboratory
One Cyclotron Road, Mail Stop 90-1116
Berkeley, CA 94720
e-mail: SAFinsterle@lbl.gov

ABSTRACT

Automatic differentiation (AD) is a way to accurately and efficiently compute derivatives of a function written in computer codes. We describe the procedures necessary to apply the AD method to the multiphase, multicomponent, nonisothermal flow simulator TOUGH2. In particular, we apply the AD method to the ECO2 module of the TOUGH2 code to explore a scheme for efficiently calculating the Jacobian matrix, which is required by the Newton-Raphson method for handling the nonlinearities arising at each iteration. The ECO2 module allows TOUGH2 to accurately simulate CO_2 sequestration in aquifers. The robustness and efficiency of the AD-generated derivative codes are compared to the conventional derivative computation approach based on first-order finite differences (FD). Our result with the test problem set indicates that the AD-generated derivative code could improve the convergence behavior in the linear solution step, taking less computational time to compute one linear matrix system.

INTRODUCTION

Geologic formations such as oil/gas fields, deep saline aquifers, or coal seams provide a great opportunity for the practical development of greenhouse gas storage systems. Numerical simulation tools are used in designing and analyzing field operation cases of the geologic sequestration systems, capturing complex processes involving fluid flow and changes in the physical and chemical properties of the porous medium. The governing flow equations for these systems are based on mass- and energy-balance considerations, resulting in a set of coupled partial differential equations (PDEs).

Since the analytical solution of the PDE system is very limited, various numerical approaches such as finite difference, finite volume, and finite element methods are used to obtain the solution over the discretized PDE domain. In these numerical approaches, the differential equations are reduced to a set of linear and nonlinear algebraic equations that relate all the involved primary variables at each gridblock of the discretized domain. Thus, the efficiency of a numerical simulator is determined to a large extent by the efficiency and robustness of the solution procedures for these algebraic equations. Nonlinear algebraic equations can be solved by various implementations of Newton's method as employed in the TOUGH2 code (Pruess, 1991).

In TOUGH2, Newton's method is implemented by seeking a sequence of solution increments $x_{p+1} - x_p$ for a given set of nonlinear algebraic equations $R(x) = 0$. Starting with an initial guess x_0, the Newton iteration continues until a predefined convergence tolerance is met. The algebraic equation for Newton's method is written as follows:

$$\left.\frac{\partial R}{\partial x}\right|_p (x_{p+1} - x_p) = -R(x_p) \qquad (1)$$

Here, $\partial R/\partial x$ is the Jacobian matrix of the partial derivative of the given nonlinear function R with respect to the independent primary variable x. The function R represents the residuals of the discretized governing equations in each gridblock. In the current version of TOUGH2, the first-order finite-difference (FD) method is used to compute an approximation of the required Jacobian matrix as follows:

$$\frac{\partial R}{\partial x} \cong \frac{R(x + \Delta x) - R(x)}{\Delta x} \qquad (2)$$

As indicated in our previous study (Kim and Finsterle, 2002), the accuracy of the Jacobian matrix $\partial R/\partial x$ depends on the step-size increment Δx (see Equation (2)). Different step sizes could lead to a different convergence behavior during the Newton iteration. This is because a small step size could cause round-off errors by subtracting two almost equal numbers, or a large step size could result in truncation errors as the omitted higher-order terms in the FD approximation become significant. Furthermore, if highly nonlinear functions are involved,

these errors can lead to a breakdown of the Newton iteration.

As an alternative to the FD method for computing partial derivatives, we have implemented automatic differentiation (AD) techniques into TOUGH2 to provide analytically computed Jacobian matrix elements. We implemented the AD approach into the ECO2 module of TOUGH2. This module was developed by Pruess and Garcia (2002) for the simulation of CO_2 injection systems in deep saline aquifers, with fluid properties for mixtures of water and supercritical CO_2 in the pressure and temperature range of 75 bars < P < 400 bars and T > 35 °C.

In the following sections, we briefly describe the AD method, after which we discuss its implementation into TOUGH2/ECO2. We then report on numerical benchmark experiments comparing AD- and FD-based versions of the code, before we conclude with a brief outline of future work.

AUTOMATIC DIFFERENTIATION (AD)

The AD method is based on computer algebra, which is applied to calculate partial derivatives of a coded function. For example, if a function *R* is computed through the elementary functional operations of *y(x)* and *z(x)*, the chain rule can be applied to compute the partial derivative of function *R* with respect to the independent variable *x* as follows:

$$\frac{\partial R\{y(x),z(x)\}}{\partial x} = \frac{\partial R}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial R}{\partial z}\frac{\partial z}{\partial x} \qquad (3)$$

The elementary derivatives for elementary functions in the equation are analytically known *a priori* and can be provided in the form of a numerical library. Based on this fundamental concept of computer algebra, various implementation techniques for AD processing have been developed. A detailed overview of the methodology and applications of AD can be found in Griewank and Coliss (1991). Juedes (1991) provided an extensive survey of AD tools available to augment computer programs written in Fortran, C, or C++.

The AD tool used in this study is the ADIFOR (Automatic Differentiation of Fortran, Bischof et al., 1998), developed by Argonne National Laboratory and Rice University. ADIFOR allows the direct manipulation of a model code to generate a derivative code that can compute analytical derivatives of the input model code, along with the original model outputs. The main advantage of the AD-generated derivative code is that all the partial derivatives are computed from a single run of a model code, whereas the FD method requires additional model runs to compute one partial derivative. Furthermore, the AD approach computes derivatives with an accuracy up to the arithmetic precision of the computer (Bischof et al., 1998), since all the involved derivative computations are performed analytically. The procedure for generating a derivative code with ADIFOR is shown in Figure 1.

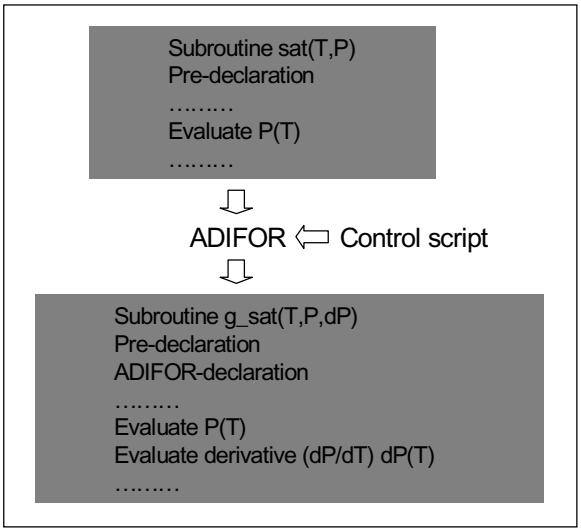

Figure 1. Schematic diagram of an automatic differentiation tool used to generate a new derivative code.

IMPLEMENTATION OF TOUGH2-AD

The governing equations of the TOUGH2 model can be written in a general form as follows (Pruess, 1991):

$$\frac{d}{dt}\int_{V_n} M^k dv = \int_{\Gamma_n} F^k \cdot n d\Gamma + \int_{V_n} q^k dv \qquad (4)$$

This equation describes the balance of mass and energy for a given arbitrary domain V_n bounded by the surface Γ_n. The quantity *M* in the accumulation term of this equation represents mass (m) or internal energy (h) per unit volume. The total mass/heat flux *F* and source/sink *q* terms are written in the right-hand side of the equation. Each mass component is labeled by *k=1,…, NK*, where *NK* is the number of components. After the surface and volume integrations are applied for each term of the above equation, a discretized form of the equation is obtained:

$$\frac{dM_n^k}{dt} = \frac{1}{V_n}\sum A_{nm} F_{nm}^k + q_n^k \qquad (5)$$

where M_n is the average value of *M* over gridblock volume V_n, A_{nm} is the interface area between gridblock *n* and *m*, and F_{nm} is the average value of the normal component of *F* over the surface segment A_{nm}. After a first-order finite-difference scheme is applied to the

time-dependent term (left-hand side) of Equation (5), the final residual equation is derived as follows:

$$R_n^{k,\,i+1} = M_n^{k,i+1} - M_n^{k,i} - \frac{\Delta t}{V_n}\left\{\sum_m A_{nm} F_{nm}^{k,i+1} + V_n q_n^{k,i+1}\right\} = 0 \qquad (6)$$

This residual equation is solved by Newton's method. As described in this paper's introduction, Newton's method requires setting up a system of linear equations consisting of the Jacobian matrix and the residual vector. Here, the Jacobian matrix is the sensitivity (or derivative) of the residual vector with respect to the selected primary variables. The typical primary variables of the TOUGH2 code are pressure, saturation, mass fraction, and temperature.

In TOUGH2, two routines construct the linear equation system, which is then solved by means of either direct or iterative linear solvers. The thermophysical properties needed to assemble Equations (3) through (6) are calculated in the ECO2 module. Key variables to be computed in the ECO2 module are indicated in Figure 2. This module is also required to provide the derivatives of the thermophysical properties with respect to the primary variables. Currently, these derivatives are computed by the FD method. The computations needed to assemble the linear equation system consisting of the residual vector and the Jacobian matrix are performed by the FLOW module. The same FLOW module is used for different applications of TOUGH2 based on different equation-of-state (EOS) modules, because the same mathematical form of Equation (1) can be applied to solve different multicomponent, multiphase flow systems, regardless of the nature and number of fluid phases and components. Only a different EOS module is needed to obtain the appropriate thermophysical properties of the fluids involved. This flexibility is a unique feature of the TOUGH2 code—a flexibility also maintained when implementing the AD method into TOUGH2/ECO2.

Many subroutines are used in the ECO2 and FLOW modules to perform the necessary calculations. To process the code with AD, the simplest approach is to combine all of these subroutines into one code and then generate one derivative code in a single AD processing step. However, mixing all the involved subroutines into one code will result in a loss of flexibility by breaking up the modular structure of TOUGH2. To avoid this, we implemented the AD method separately for each involved subroutine. After processing each subroutine with AD, we used the chain rule to combine derivative information from each routine, arriving at the final Jacobian matrix elements ($\partial R/\partial x_j$) as follows:

$$\left.\frac{\partial R_n^{k,i+1}}{\partial x_i}\right|_p = \sum_{j=1,8+NK} \frac{\partial R_n^{k,i+1}}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i} \qquad (12)$$

Here, x_i indicates each independent primary variable; the thermophysical variables computed in the ECO2 module are denoted by y_j (see Figure 2). The term $\partial y_j/\partial x_i$ indicates the derivative values of the thermophysical variables with regard to the primary variables. These derivatives are computed analytically by a new ADIFOR-processed ECO2 code. The derivative information computed by the new ECO2 module is combined with the derivative term of the residual vector $\partial R/\partial y_j$ of Equation (12). This is the term for the sensitivity data of the residual equation (6) with regard to the thermophysical variables. The new routines required to analytically compute the necessary sensitivity term $\partial R/\partial y_j$ are added to the FLOW module. A schematic diagram describing ADIFOR processing of the two key TOUGH2 modules is shown in Figure 3.

Primary variables	
Pressure	P
Temperature	T
Saturation	S

⇒

Secondary variables	
Phase saturation	S
Relative permeability	kr
Viscosity	μ
Density	ρ
Specific enthalpy	h
Capillary pressure	Pc
Diffusion factor 1	a
Diffusion factor 2	b
Mass fraction 1	x^1
Mass fraction NK	x^{NK}

Figure 2. Thermodynamic properties of the TOUGH2 module ECO2 for modeling CO_2 injection into saline aquifer systems

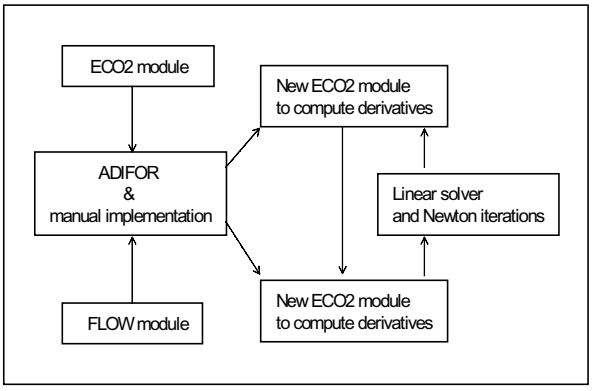

Figure 3. Automatic differentiation of TOUGH2 equation-of-state and flow modules using ADIFOR

NUMERICAL EXMPERIMENTS

A simple one-dimensional radial flow problem (see Figure 4) was chosen to check the correct implementation of the AD method into TOUGH2. All the test runs were performed in isothermal mode at T = 45°C and initial pressure of P = 120 bars. CO_2 was injected into the first element at a rate of 100 kg/s, with a constant enthalpy of 53.8 kJ/kg. Initially, the domain was fully saturated with water.

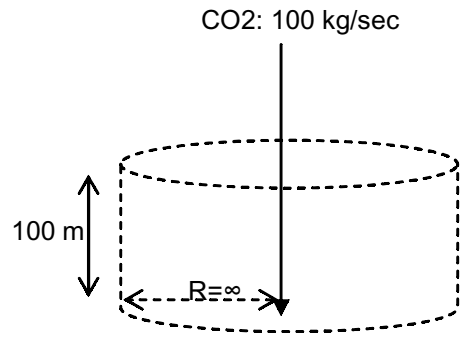

Figure 4. Schematic domain of the test problem

Table 1 shows some elements of the Jacobian matrix computed by both AD and FD methods at the first Newton iteration. The FD derivatives were computed using increment factors of 10^{-7} and 10^{-9}. The AD-computed derivatives match well with the FD-computed derivatives, especially with an increment factor of 10^{-9}. However, we observed that the FD method with the larger increment factor of 10^{-7} resulted in derivatives containing serious truncation errors. Note that the current default increment factor in the TOUGH2 code is 10^{-7}.

Table 1. Jacobian matrix elements calculated with AD and FD derivative codes

Matrix element (i,j)	AD	FD (•= 1.E-7)	FD (•= 1.E-9)
1, 1	-2.62991E-06	-2.62944E-06	-2.62944E-06
1, 3	9.05933E+03	8.82778E+03	9.05757E+03
2, 1	-1.78258E-07	-1.78229E-07	-1.78229E-07
2, 3	5.70511E+02	5.55929E+02	5.70400E+02
3, 3	4.64261E+02	4.50487E+02	4.64158E+02
4, 1	8.71731E-07	8.71583E-07	8.71583E-07
4, 3	-2.84286E+03	-2.76945E+03	-2.84230E+03

Table 2 shows the computational running time of each derivative code for different problem sizes. The AD-processed modules perform well compared to the FD-based modules. In both codes, the EOS module took up most of the computational time. Average improvement using the AD derivative code was on the order of 30%.

Table 3 shows the number of iterations required to meet the convergence criterion for both the linear and nonlinear solution steps for different injection rates of the same test problem set. The generalized minimum residual solver (DSLUGM) was used with no Z-preconditioning step in the linear solution step. We set a reduced convergence criterion of 1.E-16 to observe the effect of the Jacobian matrix accuracy on the numbers of iterations in the linear solution step. The table shows that the convergence of the linear solver is strongly dependent on the method used to compute the Jacobian matrix. A different step size of the FD method leads to a different number of iterations in the linear solution step, caused by the different accuracy of the Jacobian matrix. In most test cases, fewer iterations were taken by the AD code. With this test problem, the number of Newton iterations were identical for both AD- and FD-based codes, probably because the test problem is relatively easy to solve with a wide convergence radius in the solution search space of the Newton method.

Table 2. Performance comparison of AD- and FD-based codes for computing one Jacobian matrix

Total elements	AD		FD		AD improvement
	EOS	FLOW	EOS	FLOW	
130	0.363	0.003	0.472	0.008	28.1%
630	1.758	0.154	2.291	0.366	28.0%
1630	4.533	0.401	5.912	0.952	28.1%
2630	7.323	0.574	9.566	1.355	27.7%

Table 3. Number of iterations for AD- and FD-based codes for linear and nonlinear solution steps

	Linear iterations	Newton iterations	Total CPU time	AD improvement
Injection rate = 100 kg/sec				
AD	57	9	25.8	
FD Δ=1.E-7	107	9	64.6	60%
FD Δ=1.E-9	53	9	32.0	19%
Injection rate = 200 kg/sec				
AD	42	9	19.1	
FD Δ=1.E-7	59	9	35.7	47%
FD Δ=1.E-9	66	9	39.8	52%
Injection rate = 400 kg/sec				
AD	73	13	33.2	
FD Δ=1.E-7	80	13	48.3	31%
FD Δ=1.E-9	100	13	60.4	45%

CONCLUSIONS AND FUTURE WORK

In this paper we describe a methodology for implementing automatic differentiation techniques into the TOUGH2/ECO2 multiphase flow simulator. Automatic differentiation provides accurate analytical derivatives for the Jacobian matrix, which is calculated to handle numerically the nonlinear behavior inherent in nonisothermal, multiphase flow problems. In the numerical experiments, we determined that the automatically generated AD code provides a faster derivative computation, with faster convergence in the subsequent linear solution steps, compared to the traditional finite-difference method. For computation of one linear matrix system, the reduction in computational running time resulting from the AD derivative code was about 28%. Furthermore, the AD approach enhances the efficiency of the linear solution step, which resulted in total computational time improvement of up to 60%.

In future work, we plan to measure the computational performance of the AD derivative code using different problems with stronger nonlinearities. In addition, we will examine how the analytical derivatives relate to the convergence behavior of the nonlinear (Newton-Raphson) iterations, as well as the solution of the iterative linear equation solvers currently implemented in TOUGH2. Finally, AD techniques will be implemented to calculate the sensitivity matrix used in the various minimization algorithms of the iTOUGH2 inverse modeling code (Finsterle, 1999).

ACKNOWLEDGMENT

This work was supported by the U.S. Department of Energy, Office of Fossil Energy, under Contract No. W-31-109ENG-38 and DE-AC03-76SF00098. The encouragement of Scott Klara and Charles Byrer and the review comments by André Unger and Keni Zhang are gratefully acknowledged.

REFERENCES

Bischof, C., A. Carle., P. Hovland, P. Khaderi, and A. Mauer, *ADIFOR 2.0 User's Guide (Revision D)*, Report ANL/MCS-TM-192, Argonne National Laboratory, Argonne, Illinois, 1998 (see also http://www.mcs.anl.gov/adifor).

Finsterle, S., *iTOUGH2 User's Guide*, Report LBNL-40040 (revised), Lawrence Berkeley National Laboratory, Berkeley, California, 1999 (see also http://www-esd.lbl.gov/iTOUGH2).

Griewank, A., and G. Corliss (eds), *Proceedings of the Workshop on Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, 315–330, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1991.

Juedes, D., "Taxonomy of Automatic Differentiation Tools," in *Proceedings of the Workshop on Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, 315–330, A. Griewank and G. Corliss (eds), Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1991.

Kim, J.G., and S. Finsterle, "Application of Automatic Differentiation for the Simulation of Nonisothermal Multiphase Flow in Geothermal Reservoirs," in *Proceedings of the Twenty-Seventh Workshop on Geothermal Reservoir Engineering*, Stanford University, Stanford, California, January 28–30, 2002 SGP-TR-171.

Pruess, K., *TOUGH2 A General-Purpose Numerical Simulator for Multiphase Fluid and Heat Flow*, Report LBL-29400, Lawrence Berkeley Laboratory, Berkeley, California, 1991 (see also http://www-esd.lbl.gov/TOUGH2).

Pruess, K., and J. Garcia, "Multiphase Flow Dynamics during CO_2 Injection into Saline Aquifers," *Environmental Geology*, Vol. 42, pp. 282–295, 2002.