PROCEEDINGS, Twenty-Seventh Workshop on Geothermal Reservoir Engineering
Stanford University, Stanford, California, January 28-30, 2002
SGP-TR-171

APPLICATION OF AUTOMATIC DIFFERENTIATION FOR THE SIMULATION OF NONISOTHERMAL, MULTIPHASE FLOW IN GEOTHERMAL RESERVOIRS

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

[1]Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439, U.S.A
e-mail: jkim@mcs.anl.gov

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

ABSTRACT

Simulation of nonisothermal, multiphase flow through fractured geothermal reservoirs involves the solution of a system of strongly nonlinear algebraic equations. The Newton-Raphson method used to solve such a nonlinear system of equations requires the evaluation of a Jacobian matrix. In this paper we discuss automatic differentiation (AD) as a method for analytically computing the Jacobian matrix of derivatives. Robustness and efficiency of the AD-generated derivative codes are compared with a conventional derivative computation approach based on first-order finite differences.

INTRODUCTION

Numerical modeling techniques play an important role in engineering geothermal field operations. Mass, momentum, and energy balance considerations result in a set of partial differential equations (PDEs) describing fluid and heat transport in fractured porous media. Analytical solutions to these PDEs are available only for very specific and limited cases. Therefore, numerical approaches (such as finite-difference, finite-element, and finite-volume methods) are often used to obtain an approximate solution at discrete points in space and time. In these numerical approaches, the differential equations are reduced to a set of linear and nonlinear algebraic equations relating all the involved primary thermodynamic variables (such as pressure, temperature, and phase saturation) to each discretized grid point. Thus, the efficiency of a numerical simulator is determined to a large extent by the efficiency and robustness of the solvers for these algebraic equations.

To solve nonlinear algebraic equations, reservoir engineers often use some variation of Newton's method. In Newton's iteration scheme, for a given set of nonlinear algebraic equations $R(x) = 0$ and a given initial guess x_0, a sequence of solution increments $x_{p+1} - x_p$ is computed until a predefined convergence tolerance is met:

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

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. Here, the function R represents the residuals of the discretized mass-balance equation for each component in each gridblock at the new time level. Conventionally, the Jacobian matrix is computed by the first-order finite difference (FD) method:

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

It is not straightforward to select an appropriate step size Δx or increment factor δ, where $\Delta x = \delta \cdot x$. Values for δ that are either too small or too large can introduce serious round-off or truncation errors. Furthermore, if highly nonlinear functions are involved, these errors can lead to a breakdown of the Newton iteration. In addition to the numerical accuracy problem, the computational cost of this method can be high for a large-sized problem, since it requires n+1 functional computations for a Jacobian matrix with n columns.

As an alternative to the FD method, automatic differentiation (AD) provides accurate and fast calculations of partial derivatives. In this paper, we describe how the AD technique can be used within the structure of an existing geothermal flow simulator

to provide an analytically computed Jacobian matrix. The selected simulator is the TOUGH2 code (Pruess, 1991), developed at the Lawrence Berkeley National Laboratory. TOUGH2 is one of the most widely used nonisothermal, multiphase flow simulators with applications in geothermal, oil, and gas reservoir engineering, nuclear waste isolation, environmental assessment and remediation, and unsaturated zone hydrology. Benchmark data comparing the performance of AD and FD methods in TOUGH2 are presented.

After a brief description of the AD technique (Section 2) and the TOUGH2 model (Section 3), we discuss the implementation scheme of TOUGH2-AD in Section 4. The numerical experiments are described in Section 5 where we evaluate each differentiation method with typical geothermal test cases. We conclude with a brief description of future work.

AUTOMATIC DIFFERENTIATION

AD is an efficient approach relying on the fact that the derivatives of a function, no matter how complicated, can be computed by repeatedly applying the chain rule of derivative calculus to the sequential elementary operations 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 to the independent variable x as follows:

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

In this equation, truncation or round-off errors are of the derivative calculation are eliminated, because all the involved partial derivatives of elementary functional operations can be computed analytically by an AD-generated code. By applying the chain rule repeatedly, we can compute analytical derivatives of any computational function, because the computer code representing the function is the composition of elementary operations. Note that AD allows augmenting any computer program written in Fortran, C, or C++ for derivative computations.

Various implementation techniques for AD processing have been developed. Juedes (1991) provided an extensive survey for available AD tools. Two basic implementation approaches are applied in AD tools, referred to as the forward and reverse modes. In forward mode, derivatives of intermediate functional values are computed with respect to the input primary parameters. It is known from the linearity of differentiation that the computational effort required in this mode is approximately the number of input parameters multiplied by the runtime and memory of the original program.

In reverse mode, AD propagates derivatives of the final result with respect to intermediate variables (or quantities), known as adjoints. The program flow is reversed to be able to keep all of the adjoints that impact the final result. Because all of the involved intermediate values must be stored or recomputed, it is difficult to estimate the storage requirement using the reverse mode of AD. Recent activities of AD research have been centered on hybrid modes to combine the best features of the forward and reverse mode.

In this study, we use the ADIFOR tool (Bischof et al., 1998) developed by Argonne National Laboratory and Rice University, which employs a hybrid forward/reverse mode approach to generating derivatives. Given a Fortran routine of function computation and a control description, of which variables correspond to independent and dependent parameters, ADIFOR produces portable Fortran code that allows the computation of the partial derivatives of the dependent variables with respect to the independent variables, as shown in Figure 1.

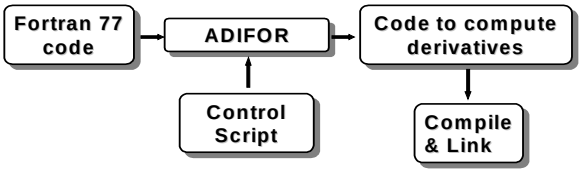

Figure. 1. Schematic diagram of the use of an automatic differentiation tool to generate a new code for derivative computations.

TOUGH2 SIMULATOR

TOUGH2 is a numerical simulator that solves the coupled equations of fluid and heat flow in a geothermal reservoir. For a given arbitrary subdomain V_n bounded by the surface Γ_n, the mass- and energy-balance equations solved in TOUGH2 model can be written in the general form (Pruess, 1991):

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

where each mass component is labeled by $k = 1,\ldots, NK$. The quantity M in the accumulation term of this equation represents mass (m) or internal energy (h) per unit reservoir volume:

$$M_m^k = \phi(S_l \rho_l x_l^k + S_g \rho_g x_g^k) \qquad (5)$$

$$M_h = \phi(S_l \rho_l u_l + S_g \rho_g u_g) + (1-\phi)\rho_R C_R T \qquad (6)$$

where ϕ is porosity, S is phase saturation, ρ is density, u is internal energy, C is specific heat, T is temperature, and x_β is the mass fraction of component k existing in each phase β. The subscripts l, g, and R indicate the phases of liquid, gas, and

rock, respectively. The total mass flux in Equation 4 is a sum of the fluxes over each phase:

$$F_m^k = \sum_{\beta=l,v} x_\beta^k F_\beta \tag{7}$$

Individual phase fluxes are derived by Darcy's law:

$$F_\beta = -\mathbf{k}\frac{k_{r\beta}}{\mu_\beta}\rho_\beta(\nabla P_\beta - \rho_\beta g) \tag{8}$$

where **k** is the absolute permeability, $k_{\beta r}$ is relative permeability, μ_β is viscosity, P_β is the pressure in the corresponding phase β, and g is the gravitational acceleration. The total heat flux is given by:

$$F_h = -\lambda\nabla T + \sum_{\beta=l,v} h_\beta F_\beta \tag{9}$$

where λ is saturation-dependent thermal conductivity, h_β is the specific enthalpy, and F_β is the mass flux of phase β. After substituting Equations 5 through 9 into Equation 1, appropriate surface and volume integration procedures are applied to obtain the discretized form

$$\frac{dM_n^k}{dt} = \frac{1}{V_n}\sum A_{nm}F_{nm}^k + q_n^k \tag{10}$$

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}. The detailed integration procedure of each term in this equation is given in Appendix B of the TOUGH2 user's manual (Pruess, 1991).

The residual equation is derived by applying the time discretization as a first-order finite difference to the left-hand-side term of Equation 10. The residual equation is written as follows:

$$R_n^{k,\,i+1} = M_n^{k,i+1} - M_n^{k,i} - \frac{\Delta t}{V_n}\left\{\sum_m A_{nm}F_{nm}^{k,i+1} + V_n q_n^{k,i+1}\right\} = 0 \tag{11}$$

This system of equations is solved by the Newton-Raphson method, as described in the introductory section above.

IMPLEMENTATION OF TOUGH2-AD

The governing equations for multiphase fluid and heat flow have the same mathematical form, regardless of the nature and number of fluid phases and components present. TOUGH2 is set up with a modular architecture, in which the main flow and transport module interfaces with an equation-of-state (EOS) module describing the thermophysical parameters of the fluids. The thermodynamic state in each gridblock is defined through a set of appropriately chosen primary variables; all other thermophysical properties needed to assemble Equation 11 are referred to as secondary parameters. The original TOUGH2 code released in 1991 provides five different EOS modules (Pruess, 1991) that can be used for different multiphase and heat transport systems. In our work for the implementation of the TOUGH2-AD model, the EOS module labeled "EOS1" was selected. The EOS1 module has been used as the most basic EOS module for geothermal applications. Two water components (water and traced water) can be handled in liquid and vapor phase under isothermal or nonisothermal conditions. The details of the formulation to describe the thermodynamic properties of water can be found in the Pruess (1991). Choice of the primary variables depends on the phase composition. Pressure and temperature are for single-phase conditions, whereas pressure and saturation are used under two-phase conditions. Primary and secondary thermodynamic variables computed in the EOS1 module are listed in Figure 2.

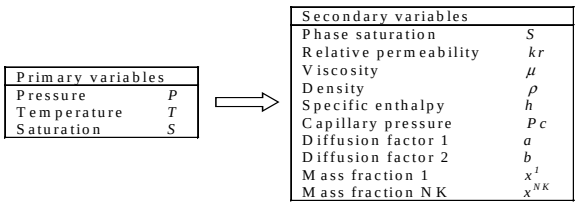

Figure. 2. Thermodynamic properties of TOUGH2 module EOS1 for nonisothermal two-phase flow of water and traced water.

Once all the thermodynamic variables are computed, the flow module calculates the accumulation and flow terms and constructs a set of linear equations (see Equation 11) for the Newton-Raphson iterations. The resulting matrix is inverted by means of either direct or iterative linear equation solvers.

To implement the AD-generated derivative codes for the TOUGH2 simulator, both the EOS and main flow module are differentiated using ADIFOR. The simplest approach for calculating derivatives of the two key modules is to combine the EOS1 and flow modules into one code and then generate one derivative code in a single AD processing step. However, mixing the two modules will result in a loss of flexibility, by breaking up the modular structure of TOUGH2. To maintain the modular structure in the AD processing, we differentiated each module separately and then combined the two AD-generated derivative codes by the chain rule to construct the Jacobian matrix comprising the linear system of residual equations. A schematic diagram describing ADIFOR processing of the two key TOUGH2 modules is shown in Figure 3.

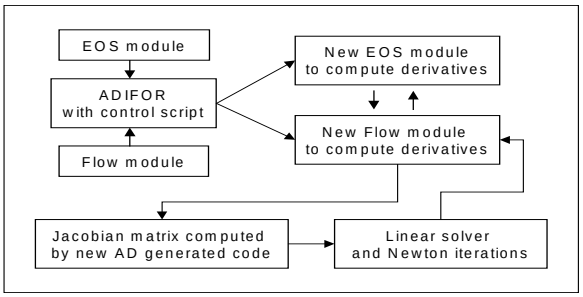

Figure. 3. Automatic differentiation of TOUGH2 equation-of-state and flow modules using ADIFOR.

Given the residual equation 11, the chain rule used to combine the two derivative codes for computing the Jacobian matrix at the *p*-th Newton iteration is written in the following form:

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

Here, x_i indicates each primary thermodynamic variable; the secondary thermodynamic variables are denoted by y_j (see Figure 2). The term $\partial y_j/\partial x_i$ is computed in a new ADIFOR-processed EOS1 code and the term $\partial R/\partial y_j$ in a new flow module generated by ADIFOR. Using this approach, the modular structure of TOUGH2 is maintained.

NUMERICAL EXPERIMENTS

We tested the AD-processed codes with typical geothermal reservoir problems. First, we compared the AD- and FD-computed derivatives to check the correct implementation of the AD method in the TOUGH2 modeling structure. A simple one-dimensional test problem consisting of 13 gridblocks was used for the comparison. Fluid was extracted from the first element of a one-dimensional, initially fully liquid saturated system. Table 1 shows some elements of the Jacobian matrix at the first Newton iteration.

Table. 1. Diagonal elements of Jacobian matrix calculated with AD and FD using different increment factors for calculating derivatives.

i,j location	AD	FD, Δ=1.E-6	FD, Δ=1.E-8	FD, Δ=1.E-10
1,1	0.482692E-06	0.482690E-06	0.482697E-06	0.477482E-06
2,2	0.373454E+07	0.373454E+07	0.373407E+07	0.371710E+07
3,3	0.482692E-06	0.482690E-06	0.482697E-06	0.477482E-06
4,4	0.373454E+07	0.373454E+07	0.373407E+07	0.371710E+07
5,5	0.144135E-06	0.144134E-06	0.144140E-06	0.143529E-06
6,6	0.235361E+07	0.235361E+07	0.235356E+07	0.235186E+07
7,7	0.144135E-06	0.144134E-06	0.144140E-06	0.143529E-06
8,8	0.235361E+07	0.235361E+07	0.235356E+07	0.235186E+07

The FD derivatives were computed using increment factors of 10^{-6}, 10^{-8}, and 10^{-10}. The AD-computed derivatives match well with the FD-computed derivatives, especially with an increment factor of 10^{-6}. However, it was observed that the FD methods with the smaller increment factor of 10^{-8} or 10^{-10} resulted in derivatives that exhibit minor, albeit inconsequential round-off errors.

We continued a similar test with a two-dimensional, five-spot, production-injection problem, which is typically encountered in deeper-zone geothermal reservoir systems (Pruess, 1991 and Pruess and Wu, 1993). This problem consisted of 180 gridblocks, with the grid pattern and the location of two wells as shown in Figure 4. Water is injected at the rate of 30 kg/s with an enthalpy of 500 kJ/kg. The same amount is produced at the upper-left corner of the domain.

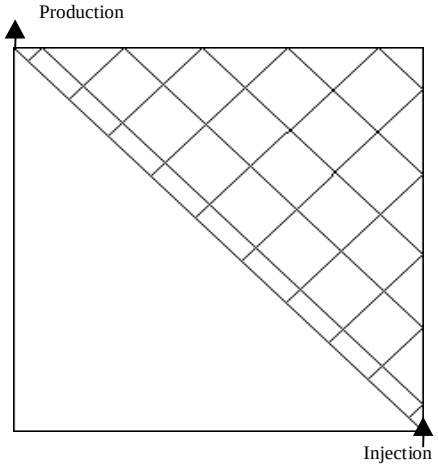

Figure. 4. The locations of injection and production wells and grid pattern (1/8 symmetric domain of a five-spot well pattern) used for test problem No. 2.

The test simulation with this problem was run for a one-week period. The derivatives calculated with AD and FD were again consistent. Minor differences between the AD- and FD-calculated derivatives are visualized in Figure 5, which shows 36×36 Jacobian sub-matrices at the first Newton iteration.

As indicated in the figure, the AD-computed and FD-computed derivatives are consistent, especially with the increment factor of 10^{-6} and 10^{-8}. With the smaller increment factor of 10^{-10}, FD results are corrupted by truncation errors, caused mainly by the inappropriate sensitivity computation of the accumulation term to the primary variables. Note that the current default increment factor in TOUGH2 is 10^{-8}. Table 2 summarizes information about the linear and nonlinear iterations observed in this test simulation. In addition, the computational running time of each derivative code is indicated.

Table. 2. Performance comparison of AD- and FD-based codes for calculating derivatives.

Methods	Linear iterations	Newton iterations	CPU time for computing Jacobian (sec)		
			Total	EOS module	FLOW module
AD	109	28	4.440	1.388	3.052
FD Δ=1.E-4	111	28	5.460	1.247	4.213
FD Δ=1.E-6	110	28	5.479	1.256	4.223
FD Δ=1.E-8	110	28	5.476	1.253	4.223
FD Δ=1.E-10	101	27	5.296	1.213	4.083

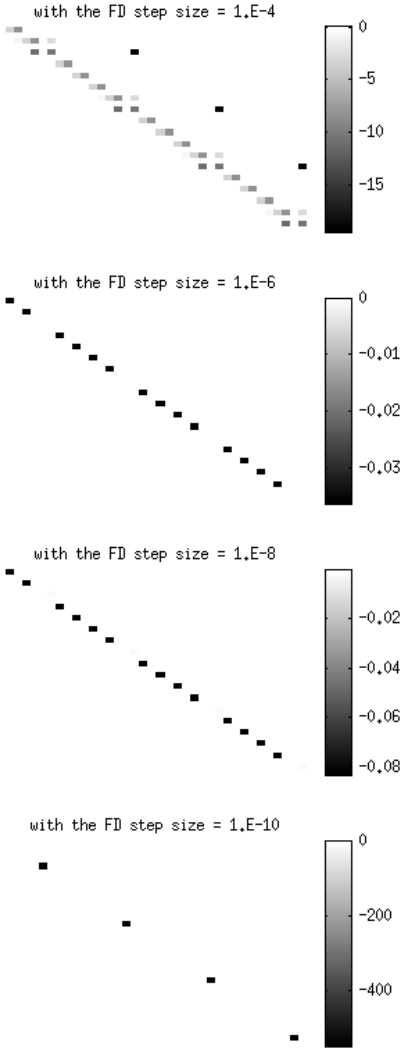

Figure 5. Derivative difference of the Jacobian matrix elements computed by the AD and FD methods.

As Table 2 indicates, the AD-processed modules perform well compared to the FD-based modules. The major computational time-saving of the AD codes is from the differentiation of the flow module, with speed-ups on the order of 20%. The AD-processed EOS module is somewhat slower than the FD code for this test case. We observed that the linear and nonlinear solution steps take almost identical iterations for the selected solver, preconditioning options, and convergence criteria.

Note that fewer Newtonian iterations were needed for the FD scheme with a increment factor of 10^{-10}, which has shown less accurate derivatives. This unexpected result could indicate that the test problem is likely to exhibit a wide (or smooth) convergence radius in the solution search space of the Newton scheme. The Newton convergence behavior of each differentiation method computed at the first time step is shown in Figure 6.

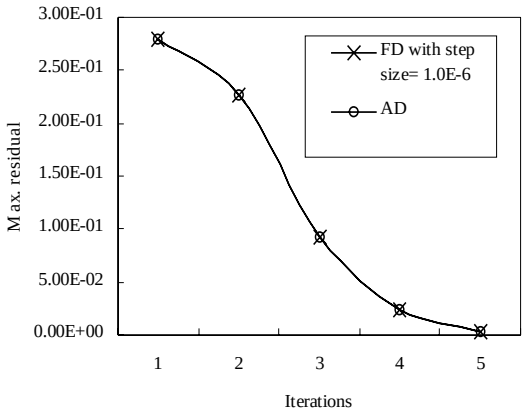

Figure. 6. Convergence behavior of AD and FD derivative codes during Newton iteration.

CONCLUSIONS AND FUTURE WORK

We described a methodology for implementing automatic differentiation techniques in the TOUGH2 multiphase flow simulator. Automatic differentiation provides accurate analytical derivatives for the Jacobian matrix, which is required to numerically capture the nonlinear behavior inherent in nonisothermal multiphase flow problems. In the numerical experiments with typical geothermal problems tractable with TOUGH2, we determined that the more accurate AD technique provides faster derivative computations compared with the conventional finite difference method. For the test cases described here, the reduction in computational running time of the AD derivative code over the FD-based derivative code was about 20%. The AD approach to calculating derivatives further enhances the efficiency and robustness of the TOUGH2 family of codes.

For future work, we plan to measure the computational performance of the AD derivative codes using large-size problems with stronger non-linearities. We expect even better performance than reported here, since the AD technique requires significantly less storage for derivative computation than the FD scheme. 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 considered for the calculation of the sensitivity matrix used in the various minimization algorithms of the iTOUGH2 inverse modeling code (Finsterle, 1999).

ACKNOWLEDGMENT

We would like to thank Dmitry Silin and Jeongkon Kim for their reviews of the manuscript. This work was supported, in part, by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Wind and Geothermal Technologies, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

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, Calif., 1999 (see also http://www-esd.lbl.gov/iTOUGH2).

Juedes, D., A taxonomy of Automatic Differentiation tools, *In*: A. Griewank 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.

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

Pruess, K., and Y.-S. Wu, A new semi-analytical method for numerical simulation of fluid and heat flow in fractured reservoirs, SPE Advanced Technology Series, Vol. 1, No. 2, 63–72, 1993.