APPLICATION OF AUTOMATIC DIFFERENTIATION TO RESERVOIR DESIGN MODELS

Amit K. Sinha[1]
Christian H. Bischof[2]

Argonne Preprint ANL/MCS-P585-0496
(revised August 1997)

To appear in ASCE Journal of Water Resources Planning and Management

ABSTRACT

Automatic differentiation is a technique for computing derivatives accurately and efficiently with minimal human effort. The calculation of derivatives of numerical models is necessary for gradient based optimization of reservoir systems to determine optimal sizes for reservoirs. We report on the use of automatic differentiation and divided difference approaches for computing derivatives for a single- and multiple-reservoir yield model. In our experiments, the ADIFOR (Automatic Differentiation of Fortran) tool is employed. The results show that, for both the single- or multiple-reservoir model, automatic differentiation computes derivatives exactly and more efficiently than the divided difference implementation. We also discuss postoptimization of the ADIFOR-generated derivative code by exploiting the model structure. We observe that the availability of exact derivatives significantly benefits the convergence of the optimization algorithm: The solution of the multireservoir problem, which took 10.5 hours with divided difference derivatives, is decreased to less than two hours with ADIFOR "out of the box" derivatives, and to less than an hour using the postoptimized ADIFOR derivative code.

KEYWORDS : Reservoir Design, Automatic Differentiation, ADIFOR, gradient-based optimization.

[1]Research Scholar, Department of Civil Engineering, Indian Institute of Technology, Bombay - 400 076, India.
[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439, USA. To whom all correspondence should be addressed.

INTRODUCTION

The mathematical relationships describing the interaction among various components of a reservoir system are, in general, complex. In a typical reservoir design formulation, these relations are often represented by mass balance equations that account for the conservation of water in the reservoir network. The physical relationships among storage, surface area of the reservoir, and reduced level of the reservoir sites are essentially nonlinear (often represented by a power function). In addition, hydropower is a nonlinear function of the storage and head available at the reservoir site. The combination of these factors eventually leads to a highly nonlinear representation of the problem, whose solution requires the application of nonlinear optimization algorithms. A nonlinear optimization algorithm employing reduced gradient search aims at the determination of a minimizer x^* to an objective function f (Gill et al. 1981),

$$f : R^n \to R,$$

using, for example, a line search technique that involves an iteration of the kind

```
for i = 1,2,…, do
    Solve $B_i s_i = -\nabla f(x_i)$
    $x_{i+1} = x_i + \alpha_i s_i$
end for
```

for suitable multipliers $a_i > 0$. Here $\nabla f(x)$ is the gradient of f at a particular point x_0, and B_i is a positive definite matrix approximating the Hessian of f. The evaluation of the gradient is a crucial ingredient of an optimization solution process. It is a rule of thumb in optimization that the accuracy of derivatives directly affects the efficiency and reliability of the optimization algorithm, where “efficiency” is measured by the number of iterations needed to obtain a good approximation to the solution (Gill et al. 1981).

Most of the applications of nonlinear programming algorithms to reservoir or hydropower optimization problems reported in the literature [e.g., Lall and Miller (1988)] have employed a divided differencing scheme to estimate derivatives. The advantage of this approach is that the function can be treated as a black box. The disadvantage is that the time required to compute derivatives grows linearly with the number of independent variables, and the accuracy of derivatives may be compromised severely as a result of truncation and cancellation errors [see, for example, Gill et al. (1981)]. To avoid these drawbacks associated with divided difference approximations, Lall (1995) presents several analytical expressions to evaluate the derivatives for a multireservoir optimization model. Derivative code can be produced from these expressions by hand and can be included in the optimization code. The advantage is that the derivatives are exact. However, the approach is tedious and requires substantial human effort for programming, debugging, and maintaining the codes. Thus, automation of this task is highly desirable.

Some early efforts in this direction include the application of symbolic differentiation programs (for example, MAPLE and Mathematica) for obtaining derivatives. Given a string describing the definition of a function, symbolic differentiation provides exact derivatives, expressing the derivatives all in terms of the independent variables. It is a powerful technique, but may run into resource limitations when the function description is complicated and may not be applicable to functions given as Fortran or C programs.

In this paper, we present the application of automatic differentiation to obtain derivatives for a complex reservoir system simulation model. Automatic differentiation is a nonapproximative method allowing fast and exact evaluation of derivatives of any degree. Given a function code and a listing of independent and dependent variables with respect to the differentiation, this method generates a derivative code that evaluates the derivative of the function. In general, depending on the particular approach chosen, automatic differentiation can compute derivatives with lower arithmetic complexity than that required by the approximate divided differences method. Furthermore, no limits are imposed on length of the program. These factors suggest that general techniques which 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 [see, e.g., (Bischof et al. 1993; Bischof et al. 1994b; Bischof et al. 1996b)].

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 (sin, cos, etc.). By applying the chain rule of differential calculus over and over again to the composition of those elementary operations, one can compute the derivative information exactly (up to machine precision) and in a completely mechanical fashion [see, e.g., (Griewank 1989)].

To illustrate, we consider a very simple example. Assume a function $f\colon x \in R^n \mapsto y \in R^m$ and that we wish to compute the derivative of y with respect to x. Here, x is called an independent variable and y the dependent variable.

$$a = x(1) + x(2) \qquad (1)$$

$$y(1) = a/x(2) \qquad (2)$$

By means of the chain rule, derivatives can be propagated forward in a mechanical fashion. Let us denote the derivatives of a variable *t* with respect to a chosen set of independent variables by *Ñt*. Then the statement (1) implies

$$\nabla a = \nabla x(1) + \nabla x(2),$$

and the chain rule, applied to the statement (2), yields

$$\nabla y(1) = \frac{\partial y(1)}{\partial a} \times \nabla a + \frac{\partial y(1)}{\partial x(2)} \times \nabla x(2),$$

which evaluates to

$$\nabla y(1) = 1.0/x(2) \times \nabla a + (-a/x(2) \times x(2))) \times \nabla x(2).$$

This mode of differentiation, where the derivatives are maintained with respect to the independent variables, is called the forward mode of automatic differentiation. The best known alternative to the forward mode is the reverse mode, which maintains derivatives of intermediate values with respect to the final results and is a discrete analog of the adjoint. The associativity of the chain rule of differential calculus actually allows many ways to compute derivatives, which may greatly differ in cost. These issues are discussed in the proceedings edited by Griewank and Corliss (1991) and Berz et al. (1996).

Various automatic differentiation tools are available. For an up-to-date account, readers are referred to the documentation available on the World Wide Web under URL http://www.mcs.anl.gov/autodiff/adtools/. In particular, we mention ADIFOR (Bischof et al. 1992, 1996b) ODYSSEE (Rostaing et al. 1993), and ADOL-F (Shiriaev et al. 1996) for Fortran programs and ADOL-C (Griewank et al. 1996) and ADIC (Bischof et al. 1996a, 1996c) for C programs. In our work, we employed the ADIFOR tool (Bischof et al. 1992, 1996b). ADIFOR differentiates programs written in Fortran 77.

THE ADIFOR AUTOMATIC DIFFERENTIATION TOOL

Given a Fortran 77 subroutine or a collection of Fortran 77 subroutines describing the function, and a designation of the dependent and independent variables with respect to differentiation, ADIFOR produces a portable Fortran 77 code that allows the computation of derivatives of the designated dependent variables with respect to the designated independent variables. ADIFOR supports almost all Fortran 77 features and, in particular, can handle arbitrary calling sequences, nested subroutines, common blocks, and equivalences. ADIFOR employs a consistent naming scheme of the subroutines and variables; therefore, the code it generates is quite amenable to postprocessing, thus allowing problem-specific postoptimization.

Experiences with ADIFOR application to problems in water resources engineering have been reported by Heidari and Moench (1997) and Bischof et. al. (1994b). For more information on ADIFOR, consult the World Wide Web at URLs http://www.mcs.anl.gov/autodiff or http://www.cs.rice.edu/~adifor.

DIFFERENTIATED MODELS: THE NONLINEAR YIELD MODEL FOR RESERVOIR DESIGN

This section presents the application and effect of automatic differentiation on the efficiency of reservoir design algorithms using the nonlinear yield model for screening multipurpose reservoir systems by Sinha et al. (1996) [see also Sinha and Rao (1996)]. A detailed description of the yield model formulation can be obtained from Loucks et al. (1981). Other reservoir sizing models include variants of linear and dynamic programming formulations in deterministic and stochastic environments [see Stedinger et al. (1983)].

These models typically use state variables (e.g., release and storage) and mass balance equations in their formulations. These models can provide an efficient way of solving the reservoir sizing problem, but their structure may lead to large formulations and, as a result, high computational requirements. Lall and Miller (1988) and Lall (1995) present a computationally compact yield model for screening reservoir systems by incorporating reservoir simulation model in a search algorithm. The yield model by Sinha et al. (1996) extends the work of Lall and Miller (1988). For the sake of completeness, a brief description of the model is given in the following sections.

The nonlinear yield model by Sinha et al. (1996) employs a hybrid simulation/optimization strategy to formulate the problem. The model differs from others (e.g., linear and dynamic programming) in the literature by not including the state variables and mass balance equations directly in the formulation. These are explicitly satisfied during the system simulation. The formulation presumes that the active storage capacity for the reservoir may be functionally evaluated, given a candidate set of releases from the reservoir. Releases are determined by applying demand fractions to the annual yield from the reservoir. A modified sequent peak algorithm is used to determine the size of the reservoir for a given sequence of release and demand. The reservoir sizing algorithm is similar to the one presented by Lall and Miller (1988) but is a direct procedure in contrast to the iterative scheme used by them. Here, the reservoir size is determined as the maximum of all minimal critical-period fillups of a hypothetical semi-infinite reservoir during its entire design period. A heuristic procedure is used to determine optimal sizes for generators required to be installed at each hydroplant [see Lall and Miller (1988)]. First, the total release possible through the hydroplant in a given month and the corresponding possible power production is computed. The possible monthly hydropower values are then treated as candidate generator sizes. The candidate that yields highest net annual revenue from hydropower generation is selected as the optimal generator size.

The decision variables are symbolically defined as

D_s	dead (conservation) storage at site *s* (reservoir site index)
I_{si}	firm annual irrigation yield from site *s* to irrigation demand zone *i*
qf_{ys}	degree (fraction) of failure of the firm irrigation yield at the site *s* for the year *y*

The objective of the model is to minimize the annualized cost of the reservoir system. The model is subject to the following constraints:
(1) *Total storage capacity* : The combination of dead storage capacity with the active storage capacity represents the total storage capacity T_s for the reservoir *s*. The total storage capacity T_s should lie between some specified upper ($T_{s,max}$) and lower ($T_{s,min}$) bound.

$$T_{s,\min} \le T_s \le \mathrm{T}_{s,\max} \quad \mathrm{s} = 1, \ldots, \mathrm{S} \tag{1}$$

(2) *Reservoir refill* : The reservoir must refill after emptying at the end of the critical period. If SF_s is the maximum active storage in the reservoir subsequent to its emptying, then the expression can be written as

$$SF_s - T_s \ge 0 \quad \mathrm{s} = 1, \ldots, \mathrm{S} \tag{2}$$

where A_s is the active storage capacity of reservoir at site *s*.

(3) *Dead storage* : The dead storage must fall within prescribed upper and lower limits. This is necessary to take care of the sedimentation requirements in the reservoir, to ensure a firm head for hydropower production, and to provide an adequate head to deliver water into the main canal.

$$DR_{s,\min} \le \frac{D_s}{T_s} \le DR_{s,\max} \quad \mathrm{s} = 1, \ldots, \mathrm{S} \tag{3}$$

where $DR_{s,min}$ and $DR_{s,max}$ are the lower and upper bound on the dead storage at ste s.

(4) *Total irrigation demand* : The irrigation demands for each demand area must be satisfied with the desired success rate (dependability). Let TI_i be the target irrigation requirement, DI_i be the lower bound for yield in dry years and $AY_{i,y}$ be the annual yield for the demand area *i* in year *y*. Then the constraint for wet and dry year yields may be written as

$$\min_y \{AY_{i,y}\} \ge DI_i \quad ; \quad \mathrm{y} \in \mathrm{N}_{\mathrm{n,f}} \tag{4}$$

$$\min_y \{SAY_y\} \ge TI_i ; \quad \mathrm{y} \in \mathrm{N}_{\mathrm{f}} \tag{5}$$

where $N_{n,f}$ is the set of years in which there is no failure of the target yield for area *i*, and N_f is the set of years in which yield failure for area *i* occurs over the simulation. The failure years are determined as part of the simulation by identifying the years for which qf_{ys} is non-zero.

(5) *System yield reliability* : This constraint is similar to the previous for demand area *s*, except that it is applied to the entire system. It is necessary to impose this constraint since the failure years for different demand areas may not be identical and therefore, even though the reservoirs may be supplying the target demand to each demand area with the prescribed dependability, the reservoir system as a whole may yet fail to meet the target irrigation demand with the desired dependability.

Let *TY* be the target system yield and *DY* be the lower bound for system yield in failure years set at 50% of the target yield. Now, consider that the array of system annual yield SAY_y values is sorted in ascending order. Then the dependability constraint may be written as

$$\min_{\mathrm{y}} \{\mathrm{SAY_y}\} \ge \mathrm{DY}; \quad \mathrm{y} \in \mathrm{NT_{n,f}} \tag{6}$$

$$\min_y \{SAY_y\} \ge TY; \quad \mathrm{y} \in \mathrm{NT_f} \tag{7}$$

where, $NT_{n,f}$ is the set of years in which there is no failure of the target yield, and NT_f is the set of years in which system yield failure occurs over the simulation. The set of failure years is identified as part of the simulation as the years in which any yield failure occurs anywhere in the system.

(6) *Firm Energy* : It is required that the system supplies the target annual hydropower for all years of simulation. The firm energy *FE* can be expressed as

$$FE = 12 \times \min_t \sum_{s=1}^{S} \left\{ PS_{s,t} \right\} \tag{8}$$

where $PS_{s,t}$ is the maximum possible power generation at site s in month t. $PS_{s,t}$ is the lesser of the optimal generator capacity determined at the end of iteration of the optimization model and the potential hydropower that can be generated for the available head and release in a given month t.

USING ADIFOR TO GENERATE DERIVATIVE CODE

The process of differentiating the function begins with the definition of the independent and dependent variables. ADIFOR (Bischof et al. 1992, 1996b) processes the entire simulation model code and performs an interprocedural dependence analysis to determine those variables that depend on the independent variables and affect the dependent ones. Variables thus identified are called *active*, and only those variables need to have derivative information associated with them. Other variables are called *passive*. The names for the derivative objects are derived by prefixing the original names of the variables with "*g_*", to generate "*g_<variable name>*". The length of a derivative object (*g_pmax_*) is equal to the maximum number of derivatives that can be computed simultaneously and, in general, is equal to the number of design parameters *(nparam)*. However, memory limitations may emply that *nparam* is higher than *g_pmax_* and then the Jacobian can be computed in a "stripmined" fashion as described, for example, in Bischof et al. (1994a). Details on the workings and use of ADIFOR 2.0 can be found in Bischof et al. (1996b) and the references therein.

ADIFOR can generate accurate derivative code for arbitrary Fortran programs, but the programming style can have a significant impact on the efficiency of the ADIFOR-generated code. We mention two issues we found to be relevant in our application:
Minimize the number of active variables: Since for each active variable a derivative object of length *g_p_* is generated, the number of active variables should be minimized. In particular, it may be advantageous to reuse variables containing temporary values.
Segregate variables in a common block: ADIFOR's dependence analysis for identifying active variables must be conservative to ensure correctness of the derivative code. Thus, if the program has active and passive variables lumped together in a common block, ADIFOR, by default, activates all the variables to ensure proper transmission of derivatives in the case that common blocks are reshaped. Obviously, such allocations can result in an unnecessarily large storage requirement. Therefore, it helps ADIFOR to generate more efficient code if the programmer isolates passive and active variables in different common blocks.

POSTOPTIMIZATION OF ADIFOR-GENERATED DERIVATIVE CODE

In the present work, it was required to conduct sensitivity analysis of the optimization model solution for different realizations of inputs to the simulation model. This meant solving the formulation a number of times for various levels of irrigation and hydropower development. Clearly, faster evaluation of derivatives (and therefore faster model solution expedites such an exercise. In this section, we present strategies applied in postoptimizing the ADIFOR-generated derivative code for our problem.

ADIFOR follows a consistent naming scheme for variables and subroutines. Consequently, the generated derivative codes are highly readable. This feature makes it possible to exploit domain-specific knowledge to produce codes that are more efficient than the ADIFOR-generated black box codes. In the following section, postoptimization strategies employed for each function are discussed. That is, while letting ADIFOR generate the bulk of the derivative code, we strategically employ our high-level knowledge of the model to improve the automatically generated code.

Release from Reservoir, $qr_{s,t}$

The Release $qr_{s,t}$ from the reservoir in a given month depends upon the irrigation demand fraction, yield, and failure fraction qf_{ys} for that month. Computation of $qr_{s,t}$ is not affected by any variation in the other independent variables. Consequently, one need not compute the gradient of release with respect to all independent variables, but only with respect to those variables that influence the release.

Active Storage Capacity Function

If *m2* is the period recording the maximum accumulated deficit, and *m1* is the first period preceding *m2* when the reservoir was at full level, then the active storage capacity can be expressed as

$$A_s = f\,(qr_{s,t},\ qi_{s,t},\ e_{s,t}\ ),\ \text{ for } t \in \{m1,...,m2\}, \tag{9}$$

where A_s is the active storage capacity of the reservoir at site s, $qr_{s,t}$ is the release from the reservoir s in period t, $qi_{s,t}$. is the inflow into site s in period t, and $e_{s,t}$ is the pan evaporation at site s in month t. This information can be used to obtain faster derivatives for the storage capacity function by computing derivatives for active $qr_{s,t}$ and the intermediate storage stat need only for the range (*m1*-*m2*) of the operation period.

Reservoir Refill Function

The refill constraint is a function of inflow, draft, and losses occurring in the range (*m2+1,m3*) of the operation period, where *m3* is the period when the reservoir refills subsequent to its emptying at the end of critical period. We express this mathematically as

$$SF_s = f\,(qr_{s,t},\ qi_{s,t},\ e_{s,t}\ ),\ \text{ for } t \in \{m2+1,...,m3\}, \tag{10}$$

where SF_s is the maximum storage content in the reservoir subsequent to the period *m2*. Therefore, notwithstanding the length of the historical period of inflow record, with this information one can obtain the derivative of the function by calculating derivatives for intermediate variables for exactly the number of months required for refilling the reservoir subsequent to its emptying.

Zonal Irrigation Demand Function

The ADIFOR-generated code computes the derivative for this function by differentiating irrigation yield functions for each year. However, this is unnecessary, since the optimization algorithm requires the derivative for the year corresponding to the "min" demand function only. Therefore, if the annual yield at irrigation zone *z* is minimum for the year $k1_z$, it suffices to calculate derivatives of intermediate variables for this year alone in order to obtain the derivative for this function.
The annual system demand and firm energy constraints are similar to the zonal irrigation demand constraint; both have the form $\min_t \{f_t\} \geq k$. Consequently, by identifying suitable "flags", these functions can be handled in a similar fashion.

Generator Capacity, G_s

The optimal generator capacity required to be installed at a site *s* is a function of potential hydropower produced in a particular month ig_s that yields maximum hydropower revenue. Therefore, it will suffice to calculate the derivatives for hydropower in the ig_s month only.

Summary of Use of post-optimized ADIFOR Code

The new process of computing derivatives can be summarized in the following steps.
- Use the original source code for function to compute its value.
- Record the value for the "*flags*" (*m2, m3,* etc.) computed as part of simulation.
- Employ the postoptimized ADIFOR-generated code to obtain the derivatives for the functions.

The foregoing discussion suggests that evaluation of derivatives is now a two-phase process. In the first phase, the function is evaluated and the information about "*flags*" is recorded. In the second phase, derivatives are computed using this information. We emphasize that, unlike the "black-box" ADIFOR-generated code, the postoptimized ADIFOR-generated code is not a stand-alone code because it cannot compute the derivatives in isolation; rather, it now depends upon the information from the generic simulation model.

The degree to which a given ADIFOR code can be postoptimized depends upon the problem structure and the extent of improvement desired by the user. Therefore, one should first profile the code to make sure that the computational savings achievable through postoptimization are worth the effort. In the present application it was quite easy to postoptimize the derivative code for the single-reservoir model to a point where it almost looked like a well written hand code, but the postoptimization of the derivative code for the multireservoir model required considerable effort. The simulation code for the multireservoir system has to account for the physical networking of the reservoirs and specific issues of riparian rights, operating policies etc. for each reservoir. This results in a complex structured code that makes postoptimization more difficult. As a result, we stopped optimizing the multireservoir code at a point where the reduction in computational time achieved was in our opinion deemed sufficient, even though the relative improvement (compared to “black-box” ADIFOR-generated code) was not as good as for the single-reservoir model.

COMPUTATIONAL RESULTS

The formulation is solved for the Par - Tapi - Narmada river system located in the western part of India. There are seven reservoirs in the system: The Jheri and Mohankavchali reservoirs are in tandem and the Paikhed, Chasmandva, Chikkar, Dabdar and Kelwan reservoirs are in parallel having independent catchments. Details of the river system can be obtained from Khaliquzzaman and Chander (1997).

In the present work, the feasible sequential quadratic programming (FSQP) algorithm (Zhou and Tits, 1992) is used to solve the yield model formulation. In this section, we present the computational results obtained from the single- (Jheri reservoir) and multiple-reservoir model using divided-difference approximations, ADIFOR “out of the box,” and postoptimized ADIFOR code to compute derivatives. All the runs were performed on a CDC 4600 workstation (approximately comparable to Sun Sparc 1+ workstation). For the multireservoir model, the 256 columns of the Jacobian were computed in a piecemeal fashion at most fourty columns at a time. Before processing with ADIFOR, the corresponding codes contained 900 and 1,500 lines of source, respectively.

Table 1 shows the CPU times and memory requirements recorded for evaluating derivatives for single- and multiple-reservoir models with different modes. For the single-reservoir model, we need to compute a *18x40* Jacobian and we observe that the computational time required by ADIFOR-generated derivative code represents an improvement by 21.1% over the divided difference implementation while the postoptimized ADIFOR-generated code reduces runtime by 97.9%. Obviously, this kind of improvement depends upon the structure of the function and may not always be possible. The size of the executable for the ADIFOR code at 1.063 Mbytes is the largest because of the derivative objects that are allocated for all active variables. Optimization of ADIFOR code decreases the number of derivative objects, resulting in an executable of 0.387 Mbytes. The divided difference implementation does not require any additional derivative objects and, with 0.354 Mbytes, requires the least amount of memory.

Similar results are shown in Table 1 for the multiple-reservoir model, which requires the computation of a *34x276* Jacobian. The code generated by ADIFOR represents a 21.6% of improvement compared with divided differences, and the optimized ADIFOR-generated code reduces derivative computation time by 99.6%. The size of the executable was 0.796 Mbytes for divided differences, 5.423 Mbytes for ADIFOR-generated code, and 1.774 Mbytes for postoptimized ADIFOR-generated code. While executables of such a size are nothing unusual, these numbers illustrate the relative memory savings that may be possible through postoptimization.

Table 2 reports on the time and memory requirements, the number of iterations, and the final value of the objective function for the solution of the optimization model for the single- and multiple-reservoir models. For the single-reservoir model, the application of automatic differentiation results in a 84.7% savings of time compared with the divided differencing scheme. The time savings further increases to 96.6% with the inclusion of postoptimization strategies in the ADIFOR-generated code. Note that the number of iterations is almost halved by having exact derivative information. The size of the executable is 0.452 Mbytes for divided differences, 1.284 Mbytes for “ADIFOR out of the box,” and 0.589 Mbytes for the postoptimized ADIFOR-generated code.

For the multiple-reservoir model, the availability of exact derivatives leads to a decrease by almost a factor five in the number of iterations, and, in addition, an improved optimal value for the objective function: 566.31 Million Rupees in contrast to the divided difference application for which the objective function value is 573.85 MRs (Million Rupees).

The results in Tables 1 and 2 illustrate the effect of additional accuracy of the derivatives obtained with automatic differentiation on the progress of FSQP algorithm. The optimization algorithm converges much faster with analytic derivatatives generated by automatic differentiation than with the divided difference implementation. As a result, the multiple-reservoir problem whose solution took roughly 10.5 hours with divided difference derivatives, was solved in less than 2 hours using ADIFOR “out of the box” derivatives, and in less than an hour with postoptimized ADIFOR derivatives.

CONCLUSIONS

Solutions to nonlinear optimization models require the evaluation of gradients of the objective function and constraints at each iteration. Conventional techniques cannot be relied upon to deliver fast and accurate derivatives for sophisticated simulation models. Divided differences may not be accurate and are obtained slowly, symbolic approaches do not appear to be feasible, and hand coding of derivatives is impractical. In contrast, automatic differentiation can be used to obtain fast and accurate derivatives for functions defined by large codes. The problems of convergence of optimization algorithm are more pronounced with large scale reservoir systems, whose faithful simulation can entail considerable coding effort.

In our experiments, we employed the ADIFOR 2.0 automatic differentiation system in the context of reservoir design models. The computational performance obtained with a "black box" application of ADIFOR can be further enhanced by postoptimizing the generated derivative code. The extent of optimization depends upon the nature of the problem, one's knowledge of the code, and the degree of improvement desired. However, even unoptimized ADIFOR-generated code performs superior to divided difference approximations, both on a "per-Jacobian" basis and overall, with the majority of savings derived from faster convergence because of more accurate derivatives. Limited (in terms of human time) postoptimization efforts yield further improvements in derivative runtime and overall, suggesting that the combination of ADIFOR and some postoptimization with human insight into the problem provides a reliable and efficient way to solve reservoir design problems.

We illustrated this approach by applying it to both a single- and multiple-reservoir problem. We described the use of ADIFOR and the postoptimizations that we employed based on our mathematical understanding of the model. We presented experimental results that document both the improved convergence of the optimization algorithm resulting from the improved accuracy of derivatives and the runtime savings achieved by using ADIFOR-generated code (unchanged or postoptimized) to compute derivatives. For the single-reservoir model, the overall runtime is reduced by 85% and 97%, respectively, whereas improvements of 82% and 91% were observed for the multireservoir model. In the latter case, computer time was reduced from 10.5 hours to less than an hour.

For large networks of reservoirs, the computational cost required for a solution with a divided differences implementation of derivatives can be prohibitive. Automatic differentiation has the potential of enabling the computation of an optimal solution to such problems at a reasonable cost, thus allowing the investigation of higher-resolution models. The improved solutions obtained by such models can be expected to contribute to better decision making in selecting an appropriate design for the system. Note that already the derivatives generated by ADIFOR "out of the box", without any human development effort, lead to substantial computational improvements. Thus, postoptimization efforts along the lines described in this paper may not be necessary unless tight computational resource bounds require further reductions in CPU time and/or memory usage.

ACKNOWLEDGMENTS

The first author thanks B. Vasudeva Rao for providing encouragement in carrying out the work. The authors are also grateful to Dmitri Shiriaev and the anonymous reviewers for helpful comments and thank André L. Tits and Jian L. Zhou for supplying a copy of the FSQP implementation. The work of the second author was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U. S. Department of Energy, under contract W-31-109-ENG-38. The field data used in the work was provided by the National Water Development Agency, New Delhi.

APPENDIX. REFERENCES

Berz, M., Bischof, C., Corliss, G., and Griewank, A. (1996). "Computational Differentiation - Techniques, Tools, and Applications." *Society for Industrial and Applied Mathematics*, Philadelphia.

Bischof, C., Carle, A., Corliss, G., Griewank, A., and Hovland, P. (1992). "ADIFOR - Generating derivative codes from Fortran programs." *Scientific Programming*, 1(1), 11-29.

Bischof, C., Corliss, G., Green, L., Griewank, A., Haigler, K., and Newman, P. (1993). "Automatic differentiation of advanced CFD codes for multidisciplinary design." *Journal on Computing Systems in Engineering*, 3(5), 625-638.

Bischof, C., Green, L., Haigler, K., and Knauff, T. (1994a). "Parallel Calculation of Sensitivity Derivatives for Aircraft Design Using Automatic Differentiation." *Proceedings 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA Paper 94-4261, American Institute of Aeronautics and Astronautics, Reston, VA.

Bischof, C., Whiffen, G., Shoemaker, C., Carle, A., and Ross, A. (1994b). "Application of automatic differentiation to groundwater transport models." Alexander Peters, Ed., *Computational Methods in Water Resources X*, Kluwer, Dordrecht, pp. 173-182.

Bischof, C., Jones, W. T., Mauer, A., and Samareh, J., and Mauer, A. (1996a). "Experiences with the Application of the ADIC automatic differentiation tool to the CSCMDO 3-D Volume Grid Generation Code." *Proceedings 34th AIAA Aerospace Sciences Meeting*, AIAA Paper 96-0716, American Institute of Aeronautics and Astronomics, Reston, VA.

Bischof, C., Carle, A., Khademi, P., and Mauer, A. (1996b). "The ADIFOR 2.0 System for the Automatic Differentiation of Fortran 77 Programs." *IEEE Computational Science & Engineering*, 3(3), 18-32.

Bischof, C., Roh, L., and Mauer, A. (1996c). "ADIC -- An Extensible Automatic Differentiation Tool for ANSI-C.", Preprint ANL/MCS-P626-1196, Argonne National Laboratory.

Gill, P. E., Murray W., and Wright M. H. (1981). *Practical Optimization.* Academic Press, New York.

Griewank, A. (1989). "On automatic differentiation." In *Mathematical Programming: Recent developments and Applications,* ed. A. L. Norwell, Kluwer Academic Publishers, Mass., 83-108.

Griewank, A. and Corliss, G. F. (1991). *Automatic Differentiation of Algorithms: Theory, Implementation, and Application.* Society of Industrial and Applied Mathematics, Philadelphia.

Griewank, A., Juedes, D., and Utke, J. (1996). "ADOL-C: A package for the automatic differentiation of algorithms written in C/C++." *ACM Transactions on Matematical. Software*, 22(2), 131-167.

Heidari, M., and Moench, A.. (1997). "Evaluation of unconfined-acquifer parameters from pumping test data by nonlinear least squares", *Journal of Hydrology*, 192, 300-313.

Khaliquzzaman and Chander, S. (1997). "Network flow programmig model for multireservoir sizing." *Journal of Water Resources Planning and Management*, ASCE, 123(1), 15-22.

Lall, U. (1995). "A yield model for screening surface and ground water development." *Journal of Water Resources Planning and Management*, ASCE, 121(1), 9-22.

Lall, U., and Miller, C. W. (1988). "An optimization model for screening multipurpose reservoir systems." *Water Resources Research*, 24(6), 953-968.

Loucks, D. P., Stedinger, J. R., and Haith, D. A. (1981). *Water Resources Systems Planning and Analysis.* Prentice-Hall, Englewood Cliffs, N.J.

Rostaing, N., Dalmas, S., and Galligo, A. (1993). "Automatic differentiation in ODYSSEE." *Tellus*, 45a(4), 558-568.

Shiriaev, D., and Griewank, A. (1996). "ADOL-F: Automatic differentiation of Fortran codes." In Berz et al. (1996), 375-384.

Sinha, A. K., and Rao, B. V. (1996). "An optimization model for screening multipurpose reservoir systems with explicit yield reliability considerations." In *Proceedings of the International Conference on Hydrology & Water Resources*, V. P. Singh and B. Kumar, Eds., Vol. 4, Kluwer Academic Publishers, Netherlands.

Sinha, A. K., Rao, B. V., and Lall, U. (1996). “A yield model for screening multipurpose reservoir systems.” Technical Report, IIT Bombay.

Stedinger, J. R., Sule, B. F. and Pei, D. (1983). “Multiple reservoir system screening models.” *Water Resources Research*, 19(6), 1383-1393.

Zhou, J. L., and Tits, A. L. (1992). *User’s guide for FSQP*. Version 3.1, Systems Research Centre, Univ. of Maryland.

APPENDIX. NOTATIONS

A_s	active storage capacity at site *s*
$AY_{i,y}$	annual yield for irrigation area *i* in the year *y*
D_s	dead or conservation storage at reservoir site *s*
DI_i	target irrigation requirement for area *i* in a deficit year
DY	target system irrigation yield in a deficit year
$e_{s,t}$	unit evaporation loss at site *s* in month *i*
FE	firm energy provided by the system
$G_{s.}$	Generator capacity at site *s*
I_{si}	firm annual irrigation yield from site *s* to irrigation demand zone *i*
$m1_s$	beginning of the critical period at site *s*
$m2_s$	end of the critical period at site *s*
qf_{ys}	degree (fraction) of failure of the irrigation yields from the site *s* in the year *y*
$qi_{s,t}$	inflow into the site *s* in month *t*
$qr_{s,t}$	release from reservoir at site *s* in month *t*
SAY_y	annual yield from the reservoir system in the year *y*
SF_s	maximum storage recorded subsequent to emptying reservoir at site *s*
T_s	total storage capacity at site *s*
TI_i	target irrigation requirement for area *i*
TY	target system irrigation yield

TABLE 1. Derivative evaluation with different derivative modes for single- and multiple-reservoir model

Derivative Mode	Size of Executable (MBytes)	CPU Time (sec)	AD Improvement (%)
Single-reservoir Model: Jacobian size = 18x40			
Divided Differences (DD)	0.354	3.01	
Automatic Differentiation (AD)	1.063	2.36	21.1
Optimized AD (OAD)	0.387	0.01	97.9
Multiple-reservoir Model: Jacobian size = 34 x 276			
Divided Differences (DD)	0.796	152.68	
Automatic Differentiation (AD)	5.423	120.50	21.6
Optimized AD (OAD)	1.774	3.20	99.6

TABLE 2. Optimization of single- and multiple-reservoir model with different derivative modes

Derivative mode	No. of Iterations	Objective Function (MRs)	Size of Executable (MBytes)	CPU Time (sec)	AD Improvement (%)
Single-reservoir Model					
DD	162	604.86	0.452	1224.25	
AD	92	604.86	1.284	187.32	84.7
OAD	90	604.86	0.589	41.83	96.6
Multiple-reservoir Model					
Derivative Mode	No. of Iterations	Objective Function (MRs)	Size of Executable (MBytes)	CPU Time (sec)	AD Improvement (%)
DD	200	573.85	3.841	37978.53	
AD	44	566.31	6.384	6957.35	81.7
OAD	41	566.31	4.811	3525.42	90.7

Note: MRs = Million Rupees