1 of 1

Reverse Automatic Differentiation of Modular Fortran Programs*

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

J. E. Horwedel

Computing Applications Division
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6011
Oak Ridge, Tennessee 37831-6370

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

SEP 29 1993

OSTI

To be published in *Theory Institute on Combinatorial Challenges in Computational Differentiation Workshop Proceedings*, Argonne National Laboratory, May 24–26, 1993, Mathematics and Computer Science Division.

*Research sponsored by the U.S. Department of Energy, managed by Martin Marietta Energy Systems, Inc., under contract No. DE-AC05-84OR21400.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

973

REVERSE AUTOMATIC DIFFERENTIATION OF MODULAR FORTRAN PROGRAMS

J. E. Horwedel

The calculation of derivatives necessary for sensitivity analysis or for the optimal solution of systems of nonlinear equations continues to be an important research objective. Several software systems have been developed for implementing automatic differentiation of computer programs. The forward mode of automatic differentiation is efficient for calculating derivatives for a large number of dependent variables with respect to a few independent variables. As the number of independent variables increases, the computational complexity, as measured in execution time and memory requirements, renders the forward mode impractical. The reverse mode or adjoint approach is efficient for derivatives of a few dependent variables with respect to thousands of independent variables; however, available memory and disk storage generally limit the application of the reverse mode to problems with less than a few million floating point assignments. The fundamental problem with the reverse mode of automatic differentiation is that the accumulation of derivatives is required. A code that uses 3 minutes of execution time to perform 50 million floating point assignments could easily need more than one gigabyte to store the accumulated derivatives.¹⁻³

GRESS (the GRAdient Enhanced Software System) was designed to apply automatic differentiation to large-scale FORTRAN programs in the nuclear industry without requiring changes to the coding.⁴⁻⁶ GRESS provides two methods for calculating and reporting derivatives. The CHAIN option implements the forward mode of automatic differentiation. The ADGEN option incorporates the reverse mode or adjoint sensitivity analysis methods to calculate derivatives.

A modular differentiation technique (MDT) is discussed that uses both forward and reverse modes to restrict the growth of execution time and storage requirements thus extending the size of problems to which automatic differentiation can be applied. MDT is implemented using GRESS and provides a compromise between the forward mode with its computational limitations and the reverse mode with its excessive memory or storage requirements. The effectiveness of the MDT in propagating derivatives through a computer program rests on the degree of modularity in the program. Most existing large-scale FORTRAN programs do not have the degree of modularity necessary to apply MDT in an automated fashion. The approach described is to provide the basic tools to allow one to implement MDT on a module-by-module basis in an existing code or in the development phase for a new code.

A module can be considered to be any sequence of FORTRAN statements. Any module can be represented by a computational graph. As an example consider the following formula for DIST.

$$DIST = \sqrt{\left(\sum_{i=1}^4 (y_i^2) \right)^3} \quad (1)$$

A computational graph for this formula is shown in Fig. 1. The squares in the computational graph represent arithmetic assignment statements. The reverse mode of automatic differentiation requires the accumulation of derivatives for every floating point assignment that is dependent on a declared parameter.

Fig. 1. Dependency graph for DIST

In modular form the DIST formula could be coded as a FORTRAN subroutine or function with the Y array as input and DIST as the calculated result.

A computer program can be represented as a sequence of modules each with its own computational graph. Each module is assumed to have input and output. Most large FORTRAN programs are modular in design; however, common blocks provide a mechanism by which modules can share global variables that are not provided on the link to the module. When processing a module we only have to be concerned with global variables that are accessed or stored during the execution of a module. Though a single module may have thousands of common block variables, only a subset may actually be used as dependent or independent variables. Variables that are used can be determined during execution of the module. Figure 2 shows the computational graph for a computer program with global variables available to modules.

Fig. 2. Computational graph for program with common blocks

The digits on the links indicate the number of input and output variables for each module. Module A has four inputs and six outputs, B has seven inputs and four outputs, and C has seven inputs and three outputs.

MDT is designed to work with each module independently. Once a module is completed, then either forward or reverse mode is used to calculate the derivatives of the output from the module with respect to the input. Only the derivatives of the output with respect to the input need to be stored. The decision to use forward or reverse mode does not have to be made a priori; it can be determined when the module is finished by the GRESS program.

For MDT to be feasible, the number of variables on the links between modules must be small compared to the number of variables within the modules. The more modular a code system, the more effectively one could implement MDT. A module can be as simple as a subroutine or function; however, the composition of a module is arbitrary. For example, in a code that does hundreds of iterations, each iteration could be treated as a module. Though in the long term completely automating MDT is recommended, the intent in this paper is to test MDT with existing technology. To demonstrate MDT a sample problem with a main program and two subroutines was selected. Each subroutine is called per iteration in the main program. The number of iterations can be varied. Four parameters and one dependent variable are retained after each iteration. Three methods were used to process the sample problem: (1) The GRESS ADGEN option implementing reverse mode on the entire program; (2) MDT treating each iteration as a module; and (3) MDT treating each subroutine as a module. The sample problem selected is the test program provided on the GRESS distribution diskette. Of importance in this paper is that there are two subroutines and no global variables. Shown in Fig. 3 is a plot of the maximum amount of memory required to store derivatives using each method as a function of the number of iterations. Method 1 is provided for comparison, since ADGEN requires the accumulation of derivatives for every arithmetic assignment statement.

Fig. 3. Memory required to store derivatives using MDT

The results clearly demonstrate the fundamental problem with the reverse mode of automatic differentiation; that is, the memory required to store derivatives is proportional to execution time. Interestingly, method 3 also shows a linear growth, though not as steep as method 1. For this application, as the number of iterations increases, method 2 would be the most feasible in terms of memory requirements. Memory requirement using method 2 increases by 52 bytes per iteration. With an iterative code using method 2 the expected increase per iteration would be the size of the Module Frame used to link each iteration.

The results, shown in Fig. 3, demonstrate that MDT is both practical and feasible. Though the sample problem is very limited in that it does not include common blocks and does require hand intervention in identifying modules, I am very encouraged by the results. Automating the procedure so that common block variables and variables on argument lists be automatically included as dependent or independent variables is conceptually straight forward. However, having the flexibility of allowing the user to identify modules is also desirable.

The conclusion that method 2 would be best can only be made for this application. The comparison between two different implementations of MDT raises the question as to whether it would be viable to automatically process a code to determine which method would be most appropriate. Much of the information required may not be available until execution of the model. It may be more feasible to develop tools to enable the user to implement MDT in a semi-automated fashion.

REFERENCES

1. Horwedel, J. E., *Matrix Reduction Algorithms for GRESS and ADGEN*, ORNL/TM-11261, Martin Marietta Energy Systems, Inc., Oak Ridge Natl. Lab. (1989).
2. Horwedel, J. E., Raridon, R. J., and Wright, R. Q., "Automated Sensitivity Analysis of an Atmospheric Dispersion Model," *Atmospheric Environment Vol 26A, No. 9*, (1992).
3. Griewank, A., "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation," Preprint MCS-P228-0491, Mathematics and Computer Science Division, Argonne National Laboratory (1991).
4. Horwedel, J. E., Worley, B. A., Oblow, E. M., and Pin, F. G., *GRESS Version 2.0 User's Manual*, ORNL/TM-11951, Martin Marietta Energy Systems, Inc., Oak Ridge Natl. Lab. (1991).
5. Oblow, E. M., Pin, F. G., and Wright, R. Q., "Sensitivity Analysis Using Computer Calculus: A Nuclear Waste Application," *Nucl. Sci. Eng.* **94**, 46 (1986).
6. Worley, B. A., Wright, R. Q., Pin, F. G., and Harper, W. V., "Application of an Automated Procedure for Adding a Comprehensive Sensitivity Calculation Capability to the ORIGEN2 Point Depletion and Radioactivity Decay Code," *Nucl. Sci. Eng.* **94**, 180 (1986).

**DATE
FILMED**

11 / 18 / 93

END

[REDACTED]

1

1

1