Cornell Theory Center Technical Report CTC96TR253, 8/96

On the efficient methods to solve ODEs and BVPs using Automatic Differentiation *

Arun Verma†

August 12, 1996

Abstract

A large number of physical phenomena are modeled by a system of ODEs or a system of implicit ODEs. We demonstrate applicability of automatic differentiation(AD) for efficiently solving

1. Boundary value problems in ODEs and implicit ODEs.
2. Initial state and parameter estimation problems.

The impact of using AD is two fold. Firstly, efficient methods for computing the gradient vectors and Jacobian matrices have been developed using AD. Secondly the process of getting derivatives via AD is robust, more user friendly, and provides error free derivatives. Furthermore, techniques using AD have been developed which exploit structure in the user's computation, and particularly the structure we observe in boundary value problems or state/parameter estimation problems. We demonstrate by a few experiments the efficiency gained by the usage of AD in solving these problems.

1 Introduction

A large number of engineering and scientific problems are formulated by systems of differential equations. The systems can be ordinary differential equations(ODEs) [12], implicit ODEs, or more generally, differential algebraic equations(DAEs) [2].

*This research was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-90ER25013, and in part by the Advanced Computing Research Institute, a unit of the Cornell Theory Center which receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of its Corporate Research Institute.

†Computer Science Department, Cornell University, Ithaca NY 14850, verma@cs.cornell.edu

A system of ODEs is expressed as :

$$y' = F(y, t)$$

where y' and y are n-vectors, and $F : \Re^{n+1} \rightarrow \Re^n$.

A system of implicit ODEs has the form:

$$M(y, t) \cdot y' = F(y, t)$$

where $M(y, t)$ is a matrix, usually termed as mass matrix [12].

And a system of DAEs can be written as:

$$x' = F(x, y, t) \quad , \quad g(x, y) = 0$$

where $g(x, y)$ represent a set of algebraic constraints on the system.

Most approaches solving a system of DAEs [2] involve a reduction to a system of (implicit) ODEs. Hence we restrict our attention to efficient solution of (implicit) ODEs in this paper.

Automatic differentiation (AD)[8] is a chain rule based technique for evaluating the derivatives analytically (and hence without any truncation errors) with respect to input variables of functions defined by a high-level language computer program. Derivatives (or Taylor series coefficients) of any order can be computed; see ADOL-C [9] or ADIFOR [1].

In this report, we proffer AD as a useful tool to help solve a variety of problems, e.g, efficient solution of boundary value problems in (implicit) ODEs (§2), initial state and parameter estimation problem (§3), and stiff Initial Value Problems (§4). Recently there has been considerable research on developing new high order integration methods, based on the Taylor series. These methods are very promising because AD can be employed to compute error-free Taylor coefficients of any high order very effectively. A brief note on these methods and recent developments is presented in §5. To conclude, we mention some related problems in the area of dynamical systems.

The main theme of this paper is to demonstrate <u>structured</u> application of AD for the numerical solution of (implicit) ODEs. It is hoped that the material presented will used for solving variety of problems in control and dynamical systems.

2 Boundary Value Problem

Given a system of ODEs

$$y' = F(y)$$

the boundary value problem is to estimate the initial state $y(0)$, given the value of the state at a fixed time T, i.e., $y(T)$.

Note that for clarity we are restricting ourselves to systems which don't depend explicitly on time, i.e, F is a function of state variables.

2.1 Formulation of the Problem

Consider an integration method, like Euler's method, which goes through a sequence of points $y(0)$, $y_1, y_2, \ldots, y_n$. The value $y_n = \tilde{y}(T)$ is an estimate of $y(T)$, and the accuracy of the estimate will vary according to the integration method used. Clearly $y_n = \tilde{y}(T)$ is a function of $x = y(0)$.

Generally, given an integration method, y_{k+1} can be written in terms of y_k (and possibly values at other previous points): e.g., for Euler's method, we can write :

$$y_{k+1} = y_k + h \cdot F(y_k) \tag{1}$$

Here we restrict ourselves to Euler's method as the integration method, to make the forthcoming analysis simpler. Generalization of our ideas to other integration methods, including implicit methods, is straightforward in principle.

The problem of finding the initial state $x = y(0)$ can formulated as a zero finding problem of a function $g(x)$:

$$g(x) = y_n(x) - y(T) = 0$$

2.2 Solving for the Initial State

Often zero finding problems are solved by variants of Newton's method, and hence require the use of the Jacobian of $g(x)$ w.r.t. x. Let us denote the Jacobian matrix of y_n with respect to x as J. This Jacobian matrix will invariably be **dense**, since J is the product of Jacobians corresponding to each integration step :

$$J = J_n \cdot J_{n-1} \cdots J_1.$$

Here J_i is the Jacobian of y_i with respect to y_{i-1} : e.g., for Euler's method :

$$J_i = I + h \cdot F_y(y_{i-1}).$$

The matrix F_y is the Jacobian of $F(y)$ w.r.t. y.

The Jacobians J_i might be sparse individually but the product is most likely dense. So if we want to compute the Newton step by solving a linear system in J, it will be very expensive, specially for large-scale problems.

However, as shown by Coleman and Verma [3], sparsity can be preserved if we use *extended functions*. The idea is to construct an extended function, $G_E(x, y_1, y_2, \ldots, y_n)$, which is a function of input variable x as well as all the intermediate variables as follows:

$$\boxed{\begin{array}{lll}
\text{“Solve”} & \text{for} & y_1 : y_1 - x - h \cdot F(t_0, x) = 0 \\
\text{“Solve”} & \text{for} & y_2 : y_2 - y_1 - h \cdot F(t_1, y_1) = 0 \\
 & & \vdots \\
\text{“Solve”} & \text{for} & y_n : y_n - y_{n-1} - h \cdot F(t_{n-1}, y_{n-1}) = 0 \\
\text{“Solve”} & \text{for} & z : z = y_n - y(T)
\end{array}}$$

If we differentiate this function G_E w.r.t. variables $x, y_1, \ldots, y_n$, we get an extended Jacobian matrix :

$$J_E = \begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_{n-1} & I \\ & & & & I \end{pmatrix}$$

Clearly, the extended Jacobian matrix J_E is very sparse. If we solve the system shown in equation (2), then it is straightforward to show that $\Delta x = -J^{-1}g$ where J is the true Jacobian.

$$\begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_{n-1} & I \\ & & & & I \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y_1 \\ \Delta y_2 \\ \vdots \\ \Delta y_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -g(x) \end{pmatrix} \tag{2}$$

Hence the Newton step $\Delta x \equiv -J^{-1}g$ can be computed using a sparse solve involving the extended Jacobian matrix : it is not necessary to form the true Jacobian matrix.

2.3 Experiment

We performed an experiment to demonstrate the effectiveness of forming the extended Jacobian matrices. A sample ODE was constructed, and the Newton iteration was employed to compute the initial state. The time period T was chosen to be 1, and the final state $y(T)$ was chosen to be a vector of all ones. A relative comparison of execution times required by two schemes is depicted in Figure 1. MATLAB's `ode23` was used, and was modified to return the extended Jacobian matrix for the purpose of this experiment.

The sample problem constructed is as follows:

$$y(1)' = 2y(1) + y(2) - 3 \tag{3}$$

$$y(i)' = y(i-1) + y(i) + y(i+1) - 3, \quad 2 \leq i \leq n-1 \tag{4}$$

$$y(n)' = 2y(n) + y(n-1) - 3 \tag{5}$$

The solution is the initial state with all values in $y(0)$ are equal to 1, given that the starting point for $y(0)$ is chosen appropriately. The Jacobian matrix of F has a tridiagonal structure, and hence the matrices $J_i, 1 \leq i \leq n$ are all tridiagonal. So the extended Jacobian matrix is very sparse(and large). But the Jacobian matrix of y_n w.r.t. x is completely dense, since it is a product of a sequence of tridiagonal matrices. The graph in Figure 1 shows that the execution time grows quadratically when we use the dense Jacobian, versus linearly when we use the extended Jacobian.

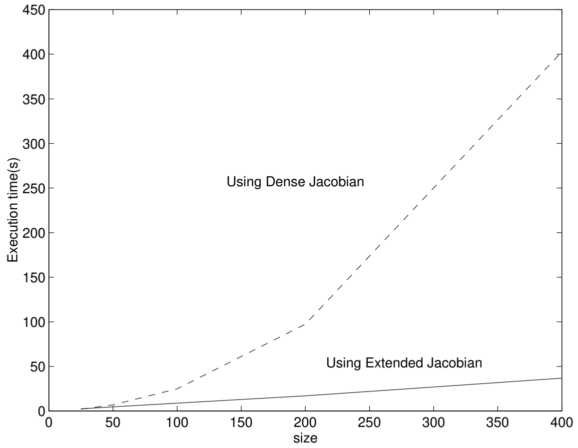

FIG. 1. *Comparison between extended Jacobian and dense Jacobian approaches*

2.4 Implicit ODE Systems

Consider an implicit ODE system :

$$M(y) \cdot y' = F(y)$$

Euler's method for this equation is :

$$y_{k+1} = y_k + h \cdot M(y_k)^{-1} \cdot F(y_k)$$

The Jacobian matrix J_i, for the mapping from y_i to y_{i+1}, is given by :

$$J_i = I + hM^{-1}(y_{i-1})F_y(y_{i-1})$$

One might fear that this representation would destroy the sparsity in the extended Jacobian matrix formulation, since M^{-1} is expected to be dense even if M is sparse. However, sparsity can be preserved, provided the Euler's method is presented in implicit form,

$$M(y_k) \cdot y_{k+1} = M(y_k) \cdot y_k + h \cdot F(y_k) \tag{6}$$

and the (hypothetical) extended function $G_E(x, y_1, y_2, \ldots, y_n)$ is constructed :

$$\boxed{\begin{array}{l}
\textit{Solve for } y_1 : M(x)y_1 - M(x)x - h \cdot F(t_0, x) = 0 \\
\textit{Solve for } y_2 : M(y_1)y_2 - M(y_1)y_1 - h \cdot F(t_1, y_1) = 0 \\
\qquad\qquad\qquad\qquad \vdots \\
\textit{Solve for } y_n : M(y_{n-1})y_n - M(y_{n-1})y_{n-1} - h \cdot F(t_{n-1}, y_{n-1}) = 0 \\
\textit{“Solve” for } z : z = y_n - y(T)
\end{array}}$$

Let us denote the Jacobian of right hand side of equation (6) by J_k, i.e.,

$$J_k = M(y_k) + hF_y(y_k).$$

If we differentiate G_E w.r.t. variables $x, y_1, \ldots, y_k$, we get the extended Jacobian matrix J_E :

$$J_E = \begin{pmatrix} -J_0 & M(x) & & & & \\ & -J_1 & M(y_1) & & & \\ & & \ddots & \ddots & & \\ & & & -J_{n-1} & M(y_{n-1}) \\ & & & & I \end{pmatrix}$$

The Extended Jacobian matrix J_E preserves the original sparsity, and hence the Newton step can be computed very efficiently.

3 State and Parameter Estimation Problem

The state and parameter estimation problems are very important problems in dynamical systems. Given a dynamical system in the form of an (implicit) ODE, the initial state estimation problem is to compute the initial state, given some measurement of the state (or observations) at certain time steps. The parameter estimation problem is to estimate a control parameter of the dynamical system given some measurement of the state (observations) at certain time steps.

In this section we present efficient schemes to numerically solve these problems.

3.1 Initial State Estimation Problem

Given the system,

$$y' = F(y)$$

the initial state estimation problem is to optimally estimate the initial state $y(0)$, given measurements on the final point :

$$\tilde{y} = H(y_n)$$

where H is generally a nonlinear function.

This problem can be formulated as a zero finding problem. To see this, consider the set of points $y_1, y_2, \ldots, y_n$ that the integration method steps through. Clearly y_n is a function of $x = y(0)$.

Now, the problem can be formulated as a zero finding problem :

$$H(y_n(x)) - \tilde{y} = 0$$

The Jacobian matrix of this function w.r.t. x is given by

$$J = H_y \cdot J_n \cdot J_{n-1} \ldots J_1.$$

where H_y denotes the Jacobian of H, and the Jacobians J_i are the same as constructed in the solution of the boundary value problem in §2. The Jacobians J_i and H_y might be sparse individually but the product will most likely be dense. So if we want to compute the Newton step using J, it will be very expensive, especially for large scale problems.

However, if we use the extended function formulation, we can preserve sparsity. The technique is to form an extended function G_E using all the intermediate variables, $y_1, y_2, \ldots, y_n$:

$$\boxed{\begin{array}{llll} \text{“Solve”} & \textit{for}\ y_1: & y_1 - (x + h \cdot F(x)) & = 0 \\ \text{“Solve”} & \textit{for}\ y_2: & y_2 - (y_1 + h \cdot F(y_1)) & = 0 \\ & & \vdots & \\ \text{“Solve”} & \textit{for}\ y_n: & y_n - (y_{n-1} + h \cdot F(y_{n-1})) & = 0 \\ \text{“Solve”} & \textit{for}\ z: & z = H(y_n) - \tilde{y} & \end{array}}$$

If we differentiate this function w.r.t. variables $x, y_1, \ldots, y_n$ we get an extended Jacobian matrix :

$$\begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_{n-1} & I \\ & & & & H_y \end{pmatrix}$$

Clearly, the extended Jacobian matrix J_E is likely to be sparse. The Newton step can be computed by solving the sparse extended Jacobian system in exactly the same way as was shown in boundary value problems in §2.

The initial state estimation can be phrased more generally. The observations can taken at a set of time instances, rather than only on the final state, i.e., optimally esimate initial state $y(0)$ given and a set of discrete measurements

$$\tilde{y}(t_j) = H(y(t_j)) \quad 1 \leq j \leq M$$

Here M is the number of measurements taken.

It is a straightforward exercise to adapt the extended Jacobian formulation for this general problem.

3.2 Initial State and Model Parameter Estimation Problem

This problem is a generalization of the initial state estimation problem. Given the system with a parameter C :

$$y' = F(y, C)$$

and observations, again with a different parameter K :

$$\tilde{y} = H(y_n, K)$$

where H is typically a nonlinear function, the problem is to optimally estimate the unknowns $y(0), C, K$.

Consider the set of points $y_1, y_2, \ldots, y_n$ that the integration method steps through. Clearly y_n is a function of $x = y(0)$ and the parameters C.

This problem can be formulated as a zero finding problem of a function $G(x, C, K)$ which is defined by :

$$G(x, C, K) = H(y_n(x, C), K) - \tilde{y} = 0$$

The Jacobian matrix of G w.r.t. x is given by

$$J_x = H_y \cdot J_n \cdot J_{n-1} \ldots J_1.$$

The Jacobian matrix of G w.r.t. C can be written

$$J_C = H_y \cdot J_n^C \cdot J_{n-1}^C \ldots J_1^C.$$

where J_k^C satisfies

$$J_k^C = hF_C(y_{k-1}, C) \tag{7}$$

The equation (7) comes from differentiating

$$y_{k+1} = y_k + hF(y_k, C)$$

w.r.t. C.

The Jacobian matrix of G w.r.t. K is given by

$$J_K = H_K$$

Here H_y denotes the Jacobian of H w.r.t. y, H_K denotes the Jacobian of H w.r.t. K. The Jacobians J_i and H_y might be sparse individually but the product is likely to be dense. So if we want to compute the Newton step using J, it will be very expensive, especially for large-scale problems.

We construct an extended function $G_E(K, C, x, y_1, y_2, \ldots, y_n)$ as follows :

$$\boxed{\begin{array}{lll}
\textit{“Solve” for } y_1 : & y_1 - (x + h \cdot F(x, C)) & = 0 \\
\textit{“Solve” for } y_2 : & y_2 - (y_1 + h \cdot F(y_1, C)) & = 0 \\
 & \vdots & \\
\textit{“Solve” for } y_n : & y_n - (y_{n-1} + h \cdot F(y_{n-1}, C)) & = 0 \\
\textit{“Solve” for } z : & z = H(y_n, K) - \tilde{y} &
\end{array}}$$

Note that K, C are now extra parameters of the extended function. If we differentiate this function w.r.t. variables $K, C, x, y_1, \ldots, y_n$ we get the extended Jacobian matrix :

$$J_E = \begin{pmatrix}
 & -J_1^C & -J_1 & I & & & \\
 & -J_2^C & & -J_2 & I & & \\
 & \vdots & & & \ddots & \ddots & \\
 & -J_{n-1}^C & & & & -J_{n-1} & I \\
H_K & & & & & & H_y
\end{pmatrix}$$

Clearly, the extended Jacobian matrix is expected to be very sparse, and the Newton step $[\Delta K \quad \Delta C \quad \Delta x]$ can be computed by solving the following extended system :

$$\begin{pmatrix} & -J_1^C & -J_1 & I & & & & \\ & -J_2^C & & -J_2 & I & & & \\ & \vdots & & & \ddots & \ddots & & \\ & -J_{n-1}^C & & & & -J_{n-1} & I \\ H_K & & & & & & H_y \end{pmatrix} \begin{pmatrix} \Delta K \\ \Delta C \\ \Delta x \\ \Delta y_1 \\ \vdots \\ \Delta y_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -g(x) \end{pmatrix}$$

The parameter and state estimation problem can be phrased more generally. The observations can be taken at a set of time instances rather than only at final time, i.e., optimally estimate $y(0), C, K$ given a set of discrete observations

$$\tilde{y}(t_j) = H(y(t_j), K) \quad 1 \leq j \leq M$$

Here M is the number of measurements taken.

It is a straightforward exercise to adapt the extended Jacobian formulation for this general problem.

4 Solving Large Stiff Systems

Many commonly occurring physical systems give rise to ODEs that are stiff [12]. Stiffness in a system is due to presence of widely varying time scales in the ODE. For a stiff ODE system given by :

$$y' = F(t, y)$$

the Jacobian matrix $J = \frac{\partial F}{\partial y}$ has a wide range of eigenvalues. This introduces a huge constraint on the step size for explicit methods, and hence implicit methods such as Backward differentiation formulas (BDF) or Rosenbrock formulas are used in practice.

The MATLAB ODE suite [10] implements BDF and Rosenbrock methods respectively in `ode15s` and `ode23s`. However, these methods require forming the Jacobian of the right hand side repeatedly. If the user doesn't provide a routine to compute the Jacobian matrix analytically, then these methods employ sparse finite differences to approximate the Jacobian matrix.

We extended the stiff solvers, `ode15s` and `ode23s` to have an option of using AD to compute the Jacobian matrices. In the rest of the section, we demonstrate the usefulness of AD in stiff solvers. In particular, we present three case studies in the next three subsections.

4.1 Ease and Effectiveness of Forming Jacobians Automatically

For most large scale problems, the Jacobian J is sparse. Recent work by Coleman and Verma [4, 3] provide techniques to exploit sparsity and structure of Jacobian matrices effectively using AD. The sparsity structure of the

Jacobian matrix can be automatically detected via AD. On the other hand, for sparse finite differencing, the sparsity structure must be provided by the user. This imposes an extra burden on the user.

In the following experiment, we demonstrate the efficiency of using AD compared to using sparse finite differences. For the purpose of demonstration we considered the following dynamical system :

$$y(1)' = -\sum_{1}^{n} y(i)^2 \tag{8}$$

$$y(j)' = -(y(1)^2 + y(j)^2) \qquad 2 \leq j \leq n \tag{9}$$

The Jacobian matrix has an arrow head sparsity structure as shown in Figure 2

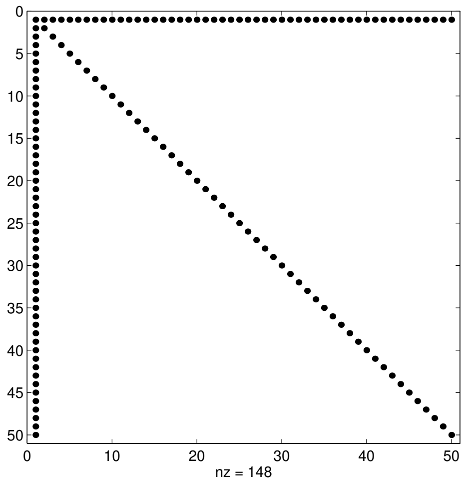

FIG. 2. *The sparsity structure of Jacobian matrix*

To compute the above Jacobian matrix of size n by finite differencing requires n function evaluations, but using the Bi-Partitioning technique of Coleman and Verma [4], which employs AD, only 3 function evaluations are required.

The timing results are tabulated in Table 1. The timings were done for `ode23s` solver in the MATLAB ODE suite.

The times required by the AD solution are much less, and the ratio increases with the problem size. However, both the methods follow almost exactly same path through integration and number of time steps required by the two are exactly the same.

4.2 Avoiding Rounding Errors due to Finite Differencing

If the function $F(t, y)$ is very sensitive to the input variables, the result you get by using finite differencing might not be close to the true result. This can affect the performance of the ODE solver, as shown in the following example :

Size	Time(FD)	Time(AD)
25	1.49	0.91
50	5.01	2.93
100	24.39	9.70
200	93.87	29.77
400	373.23	83.96
800	1145.71	253.27

TABLE 1
Comparison of AD with finite differencing approach

$$y' = -(3y^2 - y^3)$$

This is an example with a scalar variable y. The solution decays to 0 with time, if the initial condition $y(0)$ is in the interval $[0, 1]$. Assume we want to study the IVP when the starting point is very close to 0. Clearly the derivative of $F(y) = -(6y - 3y^2)$ will be accurately computed by AD, but FD will estimate :

$$F' = \frac{F(y+h) - F(y)}{h} \tag{10}$$

$$= \frac{-(h(6y - 3y^2) - h^2(3 - 3y))}{h} \tag{11}$$

$$= -(6y - 3y^2) - h(3 - 3y) \tag{12}$$

The term $h(3 - 3y)$ is the finite differencing error term. Clearly the actual derivative $-(6y - 3y^2)$ tends to 0 as y tends to 0, but the error term doesn't, assuming the finite difference step size h is fixed. Modern sparse difference routines vary h to control the error term. One intelligent choice is to keep h equal to a small fraction of function value. But then if we modify the function as :

$$y' = -M \cdot (3y^2 - y^3)$$

where M is large, say 10^5, then h will be large, and the error term will be very significant.

Here is a comparison between AD and FD, for above sample problem shown in Table 2. "Steps" denotes the number of integration steps required to integrate from $T = 0$ to $T =$"time". "f evals" records the number of function evaluations needed.

From Table 2, it can be seen that AD is superior to FD when rounding errors are important. The results above are shown for `ode23s` stiff solver.

4.3 Robust Derivatives for Nondifferentiable Functions

Consider the problem where $F(y, t)$ is a nondifferentiable function. The finite differencing method is expected to have trouble near the jumps, since the derivative approximation will be very bad.

	Using FD		Using AD	
Time	Steps	f evals	steps	f evals
0.1	50	203	25	102
0.05	47	191	22	90
0.01	40	163	15	62

TABLE 2
Comparison of AD with finite differencing approach

The particular nondifferentiable function we considered is shown in Figure 3. This function is periodic and jumps between 10 and 1 periodically.

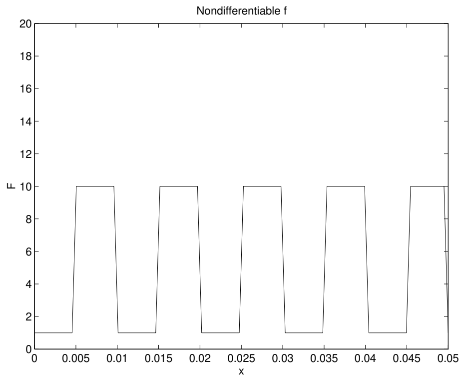

FIG. 3. *The discontinuous function*

The AD and FD solutions are plotted in Figure 4, and as expected, the FD method has problems near the corner, and it deviates away from the exact solution. AD produces the correct solution. Again the results presented the method used was `ode23s` solver.

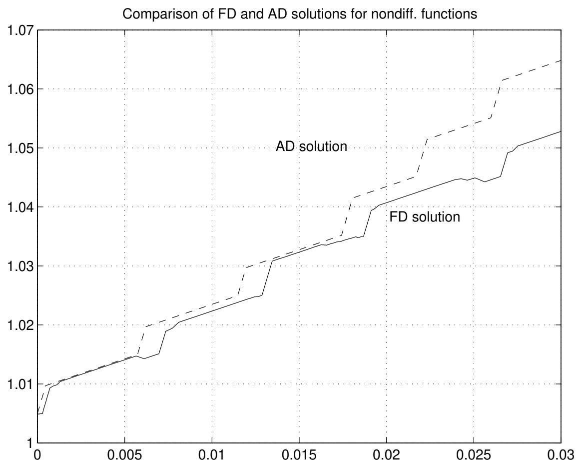

FIG. 4. *Solution Trajectories*

5 Other AD Extensions to ODE Solvers

There has been a growing interest in the AD research community about developing new integration methods based on computation of Taylor series coefficients via AD [6, 7].

Since AD can be effectively employed to compute Taylor series coefficients up to any order, researchers have tried using Taylor series methods with the aim of achieving high accuracy.

In particular, Andreas Griewank [7] utilizes the Taylor coefficients for use in Hermite-Obreshkov-Padé (HOP) class of integration methods. George Corliss et al [5, 6] have developed a package, ATOMFT, which uses classical Taylor series methods for solving ODEs.

6 Conclusions

We have illustrated the promise of using AD, in a structured way to solve to solve large stiff problems, solving boundary value problems, and for state and parameter estimation problems.

Another interesting class of problems, where the ideas presented in this paper are directly applicable, are numerical methods for DAEs.

Viable computational approaches to many other problems in dynamical systems follow from ideas presented in this paper. For example, solution of fixed points of Poincaré maps [11] can be computed by phrasing the problem as a minor variation of boundary value problem. Also, we are interested in computing the stability of the Fixed point efficiently. The stability of the fixed point is directly related to eigenvalues of the Jacobian matrix of Poincaré map. As we mentioned earlier, the true Jacobian matrix might be dense, so an interesting problem is to compute the eigenvalues of the true Jacobian from extended (sparse) Jacobian.

7 Acknowledgements

I sincerely thank Thomas F Coleman for many invaluable discussions concerning this work.

References

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

[2] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical solution of Initial value problem in Differential-Algebraic equations*, North Holland, 1989.

[3] T. F. Coleman and A. Verma, *Structure and efficient Jacobian calculation*, Tech. Rep. CTC96TR238, Theory Center, Cornell University, 1996.

[4] T. F. Coleman and A. Verma, *The efficient computation of sparse Jacobian matrices using automatic differentiation*, Tech. Rep. TR95-1557, Computer Science Department, Cornell University, November, 1995.

[5] G. F. Corliss and Y. F. Chang, *ATOMFT user manual, ver 3.11.*

[6] ——, *Solving ordinary differential equations using Taylor series*, ACM Trans. Math. Software, 8 (1982), pp. 114–144.

[7] A. Griewank, *ODE solving via automatic differentiation and rational prediction*, to appear.

[8] A. Griewank, *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers, 1989, pp. 83–108.

[9] A. Griewank, D. Juedes, J. Srinivasan, and C. Tyner, *ADOL-C, a package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, (to appear). Also appeared as Preprint MCS–P180–1190, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., November 1990.

[10] L. F. Shampine and M. W. Reichelt, *The MATLAB ODE suite*, Users Documentation, (1995).

[11] S. H. Strogatz, *Nonlinear Dynamics and Chaos*, Addison Wisley, 1994.

[12] G. Wanner and E. Hairer, *Solving Ordinary Differential equations II*, Springer-Verlag, Berlin, 1991.