Automatic Differentiation and Implicit Differential Equations*

Stephen L. Campbell † Richard Hollenbeck †

Abstract

Many physical processes are most naturally and easily modeled as mixed systems of differential and algebraic equations (DAEs). There has been an increased interest in several areas in exploiting the advantages of working directly with these implicit models. Differentiation plays an important role in both the analysis and numerical solution of DAEs. Automatic differentiation can have a significant impact on what is considered a practical approach and what types of problems can be solved. However, working with DAEs places special demands on automatic differentiation codes. More is required than just computing a gradient quickly.

This paper will begin with a brief introduction to DAEs and how differentiation is important when working with DAEs. Then the requirements in terms of both information and performance that DAEs make of automatic differentiation software will be presented. Some of our own experience in using automatic differentiation software will be mentioned. It will be seen that automatic differentiation software has a significant role to play in the future for DAEs but that not all of the demands that the numerical solution of DAEs places on automatic differentiation software are currently being met.

1 Introduction

The calculation of derivatives is a fundamental problem that arises in many different applications. A variety of software approaches including differencing, symbolic programs, and automatic differentiation have been introduced and used successfully. In this paper we will discuss our experience in using automatic differentiation software as an aid in numerically solving general higher index nonlinear differential algebraic equations (DAEs).

Unlike many applications where other computational alternatives are present, we shall see that for our application, differencing and other numerical approaches are difficult to safely implement and one is sometimes forced to turn to automatic differentiation or symbolic approaches. As will be seen, our application has other special features. Automatic differentiation (AD) is essential for doing large or complex problems. However, we need much more than just some Taylor coefficients. Many current automatic differentiation codes do not readily provide the kind of information that we require.

This paper is a continuation of our earlier paper [10] where we briefly discussed the use of symbolic and automatic differentiation software and DAEs. This paper differs from [10] is several ways. Most importantly, [10] was aimed at control engineers and was intended to show how certain information they need could be quickly computed. One of

*Research supported in part by the National Science Foundation under DMS-9423705 and ECS-9500589.

†Department of Mathematics, North Carolina State University, Raleigh, NC

the intended audiences of this paper is the automatic differentiation community. Here we discuss improvements and changes that we would like to see in available software. Another intended audience of this paper is the numerical analysis community that works with DAEs. Another difference in this paper, besides the fact the calculations are done on more modern work stations, is that we consider here larger and more complex examples arising from applications.

Section 2 develops some very basic background on DAEs including a definition of the index of a DAE. Section 3 introduces the numerical techniques which require the use of automatic and symbolic differentiation programs in order to compute the quantities defined there as J and G. A few preliminary comments on automatic differentiation are given in Section 4 including why we choose to use the code ADOL-C. How J and G are computed with ADOL-C is discussed in Section 5. For purposes of comparison, and to also serve as a check on the accuracy of our automatic differentiation results, we have symbolic software written in MAPLE for computing J and G. This is briefly described in Section 6. Several numerical studies are presented in Section 7. The first example is the space shuttle reentry problem from [1]. This is an index 3 DAE in 7 state variables. This was the largest and most complex problem looked at in [10]. The second example is an index 5 path control problem in 8 state variables for a robotic arm. Then two chemical process control problems are examined. The first is an index 3 problem in 18 state variables. The second is an index 2 problem in 58 state variables.

Calculations similar to those in this paper also appear in certain areas of control theory but we shall not discuss that relationship here.

2 General DAE Background

Differential algebraic equations (DAEs) are systems of differential equations

$$F(x', x, t) = 0 \tag{1}$$

with $\partial F/\partial x'$ identically singular. The name arises since often (1) is a mix of differential and algebraic equations. DAEs arise naturally in many areas [1], [15]. The algebraic equations can arise because of physical constraints, desired behavior, or when discretizing spatial operators during the method of lines solution of a PDE. Depending on the area of application, DAEs are also called implicit, descriptor or singular.

2.1 Analytical

Consider the following simple example of (1):

$$x_2' = x_1 \tag{2a}$$

$$x_2 = t + \alpha(t)x_3 \tag{2b}$$

$$x_3' = x_3 + 1 \tag{2c}$$

Here $\alpha(t)$ is a nonzero coefficient. The solution of (2) is

$$x = \begin{bmatrix} 1 + \alpha'(t)(-1 + ce^t) + c\alpha(t)e^t \\ t + \alpha(t)(-1 + ce^t) \\ -1 + ce^t \end{bmatrix}, \quad c \text{ an arbitrary constant} \tag{3}$$

From this example we can make several observations about how DAEs differ from ordinary differential equations.

1. The solution x of (1) can depend on derivatives of the defining equations F. Note the $\alpha'(t)$ term that appears in the solution (3).
2. Only some initial conditions will admit smooth solutions. These are called *consistent initial conditions*.
3. There can be hidden constraints. The solutions of (2) satisfy not only the constraint (2b) but also the constraint $\alpha'(t)x_3 + \alpha(t)(1 + x_3) - x_1 = 0$.
4. The best that can be hoped for is that the solutions form a smooth manifold, called the *solution manifold*. It is parameterized by t, c in the example above.

Suppose the DAE (1) is a system of n equations in the n dimensional state variable x and that F is sufficiently differentiable in the variables (x', x, t) so that all needed differentiations can be carried out. Suppose also that the solutions form a smooth manifold and that a (consistent) initial value on the manifold uniquely determines a solution. Such a DAE is sometimes called *solvable*. As noted, the solution x of (1) will depend, in general, on derivatives of F. If the ith equation of (1) is differentiated r_i times with respect to t for $i = 1, \ldots, n$, we get the $m = n + \sum_{i=1}^{n} r_i$ *derivative array equations*

$$G(x', w, x, t) = \begin{bmatrix} F(x', x, t) \\ \frac{d}{dt}F(x', x, t) \\ \vdots \\ \frac{d^r}{dt^r}F(x', x, t) \end{bmatrix} = \begin{bmatrix} F(x', x, t) \\ F_{x'}x'' + F_x x' + F_t \\ \vdots \\ F_{x'}x^{(r+1)} + \cdots \end{bmatrix} = 0 \tag{4}$$

where $w = [x^{(2)}, \ldots, x^{(r+1)}]$, $r = \max_{1 \le i \le n} r_i$. In (4) we have taken $r_i = r$ to simplify our notation.

There are several versions of the *index* of a DAE and they are not equivalent for general nonlinear DAEs [6]. For our purposes, we shall define the index ν to be the least value of r for which the derivative array equations (4) uniquely determine x' given a consistent (x, t). This index is sometimes called the differentiation index. It should be noted that in a physical model the index is not determined just by the equations but also by which variables are considered known, such as inputs, and which are considered unknown.

There are several approaches for solving DAEs using differentiation. Most of these approaches assume the DAE has some special structure. We focus here on one particular approach designed for general nonlinear DAEs.

2.2 Numerical

In order to integrate a DAE, or for that matter an ODE, we need an estimate of x'. The index measures, in some sense, how hard it is to get x'. The index also measures the loss of smoothness in going from the coefficients and forcing functions to the solutions. Finally the index also affects the conditioning of the matrices which occur in some methods such as BDF (Backward differentiation formulas). Several numerical methods have been proposed for solving DAEs including BDF or implicit Runge-Kutta (IRK) methods [1], [15]. There are also several methods designed specifically for specific applications such as constrained mechanics or electrical circuits. These approaches have proven very useful and the availability of codes has encouraged a wider consideration of DAE models. However, these methods are limited to problems of low index and special structure. For index one DAEs we have codes such as DASSL or DASPK [1]. There is no general code currently available for even index two problems.

3 General Numerical Integrators

In [2] we outlined a more general procedure that in principle can be used to numerically solve DAEs not solvable by classical means and to compute consistent initial conditions when classical methods are applicable. It is expected that this procedure will be most useful in the early stages of design and simulation. At this stage one wants to avoid time consuming manipulation of a mathematical model since simulations will be done for a variety of parameter values and altered physical configurations. The procedure is being designed for situations where the model is an implicit nonlinear solvable initial value problem of moderate size which may have no particular structure.

We assume there is a simply connected manifold of solutions. We do not assume that $F_{x'}$ has constant rank nor that its nullspace depends only on t as many approaches do.

Given a consistent value of (x,t), the derivative array equation (4), viewed as an algebraic equation, will generally have a manifold of solutions for (x',w). Suppose, however, that (4) uniquely determines x' if *consistent* (x,t) are given. That is, $r \geq \nu$. Then x' is a function of just (x,t) so that $x' = g(x,t)$. One can then integrate $x' = g(x,t)$. A detailed discussion of the technical issues with this type of approach, can be found in [2], [4], [8], [9]. It is important to note that while $G = 0$ may uniquely determine x', that is not the case with some of the higher derivatives unless extra differentiation is done.

For our purposes here it suffices to say that there is a set of assumptions which are almost equivalent to solvability. When these assumptions hold, we can try and solve $G = 0$ by minimization or some other equation solving procedure. We have investigated solving $G(z) = 0$ using a Gauss-Newton iteration

$$z_{n+1} = z_n - \rho_n G'(z_n)^\dagger G(z_n) \tag{5}$$

where $G'(c)^\dagger G(d)$ is the minimum norm least squares solution of $G'(c)z = G(d)$ and ρ_n is a damping factor to increase the region of convergence of the iteration.

In the actual numerical procedures we are working on, the Gauss-Newton iteration takes several different forms. It is used for computing consistent initial equations. It is also used to generate an ODE, called a completion, which includes the solutions of the DAE and can be integrated by standard integrators. For either integration or initialization we sometimes know all or part of x. These variables are then held constant and z is x', w and perhaps part of x.

Like all numerical methods we try to reduce the computational cost. In some cases we may reuse Jacobians so that the c and d in $G'(c)^\dagger G(d)$ need not be the same.

Regardless of which particular computation we are performing we need to be able to

1. Compute G given values of x, t, w.

2. Compute the Jacobian $J = [\ G_{x'} \quad G_w \quad G_x\ ]$.

The computation of G and J will have to be done a large number of times since we must solve (4) one or more times at every time step as the integration of the DAE progresses.

As a very simple example of G and J suppose that the DAE is

$$x_1' + x_1 x_2 = 0 \tag{6a}$$

$$x_1^2 + x_1 x_2 + 1 = 0 \tag{6b}$$

and that $r = 2$. Then

$$G = \begin{bmatrix} x_1' + x_1 x_2 \\ x_1^2 + x_1 x_2 + 1 \\ x_1'' + x_1' x_2 + x_1 x_2' \\ 2x_1 x_1' + x_1' x_2 + x_1 x_2' \\ x_1''' + x_1'' x_2 + 2x_1' x_2' + x_1 x_2'' \\ 2x_1 x_1'' + 2(x_1')^2 + x_1'' x_2 + 2x_1' x_2' + x_1 x_2'' \end{bmatrix}$$

In the notation of (4) we now have $x = [x_1, x_2]$, $x' = [x_1', x_2']$ and $w = [x_1'', x_2'', x_1''', x_2''']$. Thus

$$[G_{x'}\ G_w\ G_x] = \left[\begin{array}{cc|cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 & x_2 & x_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2x_1 + x_2 & x_1 \\ x_2 & x_1 & 1 & 0 & 0 & 0 & x_2' & x_1' \\ 2x_1 + x_2 & x_1 & 0 & 0 & 0 & 0 & 2x_1' + x_2' & x_1' \\ 2x_2' & 2x_1' & x_2 & x_1 & 1 & 0 & x_2'' & x_1'' \\ 4x_1' + 2x_2' & 2x_1' & 2x_1 + x_2 & x_1 & 0 & 0 & 2x_1'' + x_2'' & x_1'' \end{array}\right]$$

In our algorithms we must frequently deal with matrices where a rank determination is necessary. For example, in the ICP approach the condition number of a submatrix is used to decide when it is necessary to compute new coordinate partitions. This, combined with the need to compute higher derivatives, makes differencing impractical. We are forced to consider either symbolic or automatic differentiation software for computing G and J. There are several automatic differentiation codes that can compute G. The need to also compute J severely restricts our choices. We chose ADOL-C since it was the only AD code we knew of that could compute what we needed.

4 Automatic Differentiation

Automatic differentiation is based on the observation that all functions are evaluated as composites of arithmetic operations and a finite library of nonlinear functions. Through these elementary functions, derivatives of arbitrary order can be propagated in the form of truncated Taylor series. In AD this propagation is numeric and not symbolic. No truncation error occurs in this straight forward application of the chain rule. We shall write our discussion in notation consistent with the output of ADOL-C and with an eye toward our particular application to differential algebraic equations.

Suppose that we have a scalar function

$$u = Q(h) \tag{7}$$

where we think of h as a function of the independent variable t. At a fixed value $\widetilde{t}$ of t, h will have a power series in terms of $\epsilon = t - \widetilde{t}$. Similarly, u will have a power series in ϵ. Let $f^{(i)}(t)$ denote the i-th derivative of a function f with respect to t and set $f_i(\widetilde{t}) = f^{(i)}(\widetilde{t})(i!)^{-1}$ so that the f_i are Taylor coefficients. Thus

$$\sum_{i=0}^{p} u_i \epsilon^i = Q(\sum_{i=0}^{p} h_i \epsilon^i) \tag{8}$$

We have chosen to write (8) in truncated Taylor series though that does not matter as long as p is greater than the desired derivatives. In [19] the vector $[f(\widetilde{t}), f'(\widetilde{t}), \ldots, f^{(i)}(\widetilde{t})]$ is called a dynamic form.

The coefficients u_i depend on the coefficients h_j of h. ADOL-C computes the Taylor coefficients u_i and the partial derivatives

$$\frac{\partial u_i}{\partial h_j} \tag{9}$$

Note that (9) is the partial derivative of a Taylor coefficient with respect to a Taylor coefficient. At this time most AD codes only return the u_i. The need for computing (9) was a major factor in choosing ADOL-C.

It will be useful later to note that [11]

$$\frac{\partial u_i}{\partial h_j} = 0 \quad \text{if } j > i \tag{10}$$

$$\frac{\partial u_i}{\partial h_i} = Q'(h) = Q'(h_0) \tag{11}$$

$$\frac{\partial u_i}{\partial h_j} = \frac{\partial u_{i-k}}{\partial h_{j-k}} \quad \text{for } i \geq j \geq k \geq 0 \tag{12}$$

There are two phases to an automatic differentiation code like ADOL-C. In the forward mode, the u_i of (8) are computed and the information on variable dependencies is stored in a file called a "tape" in ADOL-C. In the reverse mode the partial derivatives (9) are computed using this tape.

5 G and J using Automatic Differentiation

We now describe how the automatic differentiation program ADOL-C is used to compute G and J at current values of t, x, v, w. It suffices to consider an autonomous DAE. Let

$$y = F(x', x) \tag{13}$$

For simplicity we differentiate all equations the same amount. Then at time $\widetilde{t}$,

$$G(x', w, x) = \begin{bmatrix} F(x', x) \\ \frac{d}{dt}F(x', x) \\ \vdots \\ \frac{d^r}{dt^r}F(x', x) \end{bmatrix} = \begin{bmatrix} F(x', x) \\ F_{x'}x'' + F_x x' \\ \vdots \end{bmatrix} = \begin{bmatrix} y(\widetilde{t}) \\ y'(\widetilde{t}) \\ \vdots \\ y^{(r)}(\widetilde{t}) \end{bmatrix} = \begin{bmatrix} y_0(\widetilde{t}) \\ y_1(\widetilde{t}) \\ \vdots \\ r!y_r(\widetilde{t}) \end{bmatrix}$$

The y_i are directly available from the AD code as is G_x. However, for $\overline{J} = \begin{bmatrix} G_{x'} & G_w \end{bmatrix}$ we need

$$\frac{\partial y^{(i)}}{\partial x^{(j)}}, \quad 0 \leq i \leq r,\ 1 \leq j \leq r+1 \tag{14}$$

which is not directly output by the automatic differentiation code.

In order to use ADOL-C, instead of (13) we write

$$y = F(v, x) \tag{15}$$

Here v, x are considered independent so that at time $\widetilde{t}$ we have $v = \sum_{i=0}^{p} v_i \epsilon^i,\ x = \sum_{i=0}^{p} x_i \epsilon^i$. ADOL-C returns the quantities

$$\frac{\partial y_i}{\partial v_j}\ , \quad \frac{\partial y_i}{\partial x_j} \tag{16}$$

which are computed as if v_j, x_j are independent. ADOL-C actually only computes the $j = 0$ case of (16). The partial derivatives for other values of j are given by (12).

The relationship between the ADOL-C output (16) and the needed partial derivatives (14) was established in [10]. Let an overbar indicate that the partial is computed thinking of v, x as dependent. Using the fact that $v_{j-1} = jx_j$, $j \geq 1$;

$$\overline{\frac{\partial y_i}{\partial x_j}} = 0, \quad j > i+1,\ i \geq 0 \tag{17a}$$

$$\overline{\frac{\partial y_i}{\partial x_{i+1}}} = (i+1)\frac{\partial y_i}{\partial v_i} = (i+1)F_v = (i+1)\frac{\partial y_0}{\partial v_0},\ i \geq 0 \tag{17b}$$

$$\overline{\frac{\partial y_i}{\partial x_j}} = \frac{\partial y_i}{\partial v_{j-1}} j + \frac{\partial y_i}{\partial x_j}, \quad 1 \leq j \leq i \tag{17c}$$

$$\overline{\frac{\partial y_i}{\partial x_0}} = \frac{\partial y_i}{\partial x_0}, i \geq 0 \tag{17d}$$

Now we can express the entries of J in terms of the data from automatic differentiation.

$$\frac{\partial y^{(i)}}{\partial x^{(j)}} = \overline{\frac{\partial i! y_i}{\partial j! x_j}} = \frac{i!}{j!}\overline{\frac{\partial y_i}{\partial x_j}} = \frac{i!}{j!}\left(\frac{\partial y_i}{\partial v_{j-1}} j + \frac{\partial y_i}{\partial x_j}\right), \quad 1 \leq j \leq i \tag{18a}$$

$$\frac{\partial y^{(i)}}{\partial x^{(0)}} = \overline{\frac{\partial i! y_i}{\partial x_0}} = i!\frac{\partial y_i}{\partial x_0}, \quad j = 0,\ i \geq 0 \tag{18b}$$

$$\frac{\partial y^{(i)}}{\partial x^{(i+1)}} = \overline{\frac{\partial y_i}{\partial v_0}}, \quad j = i+1 \tag{18c}$$

To simplify the result, let

$$\beta_{i,j} = \begin{cases} \frac{i!}{j!} & \text{if} \quad i \geq j-1,\ j \geq 1 \\ 0 & \text{if} \quad i < j-1,\ , i \geq 0 \end{cases} \tag{19a}$$

$$V_{i,j} = \frac{\partial y_i}{\partial v_j}, \qquad X_{i,j} = \frac{\partial y_i}{\partial x_j} \tag{19b}$$

$$Y_{i,j} = \frac{1}{\beta_{i,j}}\frac{\partial y^{(i)}}{\partial x^{(j)}},\ r \geq i \geq 0,\ i+1 \geq j \geq 0 \tag{19c}$$

Note $X_{i,j}, V_{i,j}$ are provided by ADOL-C and $\beta_{i,j} Y_{i,j}$ are the desired nonzero entries of J.

Given two consecutive values of $Y_{i,j}$ on a subdiagonal we can compute the rest of the entries on that subdiagonal. Since in computing G, the expressions tend to get more complicated with increasing i, the most useful form of this recursion will be the one starting with the left most values of the subdiagonal. This result is of some independent interest since it can be used to speed up the computation of J even for a symbolic language. Analogous formulas can be derived for (1) by letting $y = F(v, x, \tau)$ with $\tau' = 1$.

LEMMA 5.1. *With the notation* (19) *the subdiagonals of J can be computed as follows.*

1. *Compute $Y_{k,1},\ Y_{k+1,2}$, $r - 1 \geq k \geq 1$.*
2. *Then for $k \geq 1,\ \ell \geq 2$:*

$$Y_{k+\ell-1,\ell} = (2-\ell)Y_{k,1} + (\ell-1)Y_{k+1,2} \tag{20}$$

6 Symbolic Computation of G and J

While it has many computational advantages, automatic differentiation is not always the easiest software to use. We have also written some code in MAPLE that will take the equations which describe a DAE, generate formulas for G, J, and then convert these formulas to FORTRAN (or C) subroutines. As noted in [10] it can take considerable computational effort for MAPLE to generate the routines. However, once the FORTRAN subroutines are generated, the overhead is paid and these FORTRAN subroutines can be as fast, or faster, than automatic differentiation on some "smaller" problems.

The symbolic approach is also helpful in that it can serve as a check on how the automatic differentiation software is doing on small to moderate sized problems.

7 Examples

In [10] we examined the shuttle problem which showed how automatic differentiation was a viable approach. We first return to that example and rerun it on somewhat faster hardware. In addition, we consider three more problems that arose in applications. These problems are all more complex either in the sense of being higher index, or having larger dimension.

In the tables that follow the CPU time is in seconds. All computations were done on the equivalent of a SUN SPARC 5. When integrating we frequently reuse the value of J. Thus it is of interest to know the time to compute just G. The tables give the time for ADOL-C to compute J and G. Timings were done by the UNIX time function. ADOL-C CPU times were computed by averaging 100 calls. For our application it is possible to utilize ADOL-C in such as manner that some of the setup, such as include statements need to be done only the first time. Thus our CPU time is representative of what the time per evaluation would be during an integration. A single call would be slightly higher. We have made no attempt to optimize the original C code describing the DAE to improve the performance of AD while the MAPLE fortran has been optimized. This reflects the built in features of each package. Finally, the cost includes the determination of all entries of J. The ADOL-C memory numbers are for average memory during the computation.

With MAPLE there is a considerable overhead in CPU time while MAPLE generates the needed equations and then produces the FORTRAN code. However, once the FORTRAN is generated, we need never return to MAPLE since all parameters are passed to the FORTRAN code. The columns labeled as evaluation of G and J by MAPLE refer to the time needed to evaluate the FORTRAN code. The time needed for MAPLE to find formulas for the derivatives and then generate the FORTRAN code is in the columns labeled "MAPLE CPU Time." The generation of the FORTRAN code usually takes longer than the symbolic differentiation.

These computations were all done with MAPLE V.3. The upcoming MAPLE V.4 will have some additional optimization of expression capabilities. These features will not alter any of our conclusions but they could somewhat reduce the MAPLE CPU Time on some applications.

Use of memory is important and one of the limiting factors with the symbolic approaches. The memory table gives the average memory requirement for ADOL-C during the evaluation of G and J. The entries labeled "nd" could not be done by MAPLE without moving to a larger memory allocation.

The variable r gives the amount of differentiation. Generally, we would only need to take r to be the same as the index. However, larger values of r are of interest. For one, they give us some information about the computational effort that would be needed for a higher index problem of similar size with similar functions. Also, there are situations

where one might want to take a higher value of r. For example, one might want x'' as well as x'. Finally, in practice it is often possible to differentiate some of the equations less than others. However, the usual graphical based algorithms can underestimate the amount of differentiation needed [7]. Our interest here is just in comparing AD and symbolic approaches. All equations have been differentiated the same number of times.

7.1 Shuttle reentry problem

The first test problem has functions which are typical of those that can arise in flight control and mechanics. This was the largest example considered in [10]. It is the trajectory prescribed path control problem for the shuttle in relative coordinates from [1]. The equations can be found in [1] or [8]. The consistent initial conditions are from [8]. It is an index three DAE in 7 variables.

CPU Times for the Shuttle Problem

	CPU time for evaluation of G and J				MAPLE CPU Time	
	Evaluation of G		Evaluation of J			
r	ADOL-C	MAPLE	ADOL-C	MAPLE	G	J
1	0.0040	0.0001	0.0082	0.0015	7.98	21.51
2	0.0046	0.0012	0.0107	0.0025	22.56	70.84
3	0.0048	0.0017	0.0128	0.0061	61.22	235.84
4	0.0052	0.0029	0.0158	0.0150	192.24	852.07
5	0.0056	0.0065	0.0193	nd	645.03	nd

Average Memory Use and FORTRAN Code Size in Kb.

	Average Memory Used				FORTRAN Code Size	
	MAPLE		ADOL-C			
r	G	J	G	J	G	J
1	815	1050	79.7	101.5	3.5	14.8
2	1046	1575	67.9	101.3	10.0	50.4
3	1357	2897	70.4	93.5	26.7	141.6
4	2034	6313	67.7	119.3	67.8	386.4
5	3745	nd	82.5	110.1	160.0	nd

7.2 Robotic arm problem

The second example is a path control problem for a two link robotic arm with a flexible joint. This problem is described in the appendix. There are 8 state variables. The DAE is index 5. The model is from [12]. The equations also appear in [7].

CPU Times for the Robot Arm Problem

	CPU time for evaluation of G and J				MAPLE CPU Time	
	Evaluation of G		Evaluation of J			
r	ADOL-C	MAPLE	ADOL-C	MAPLE	G	J
1	0.0038	0.0010	0.0081	0.0014	8.25	18.41
2	0.0041	0.0012	0.0112	0.0022	19.73	50.61
3	0.0045	0.0015	0.0143	0.0036	47.67	129.70
4	0.0048	0.0022	0.0163	0.0072	111.50	324.18
5	0.0053	0.0042	0.0202	0.0154	291.05	987.86

Average Memory used in Kilobytes.

	MAPLE		ADOL-C	
r	G	J	G	J
1	782	974	71.4	97.0
2	959	1258	78.9	87.9
3	1112	1780	66.3	91.3
4	1392	2899	69.4	116.9
5	1902	5235	65.5	109.1

7.3 Reactor Separator System

The next two examples come from chemical process control. In the first one, the process is comprised of a two-phase reactor and a condenser (separator) [17]. This is an index 3 DAE in 18 variables.

CPU Times for the Reactor Separator System

	CPU time for evaluation of G and J				MAPLE CPU Time	
	Evaluation of G		Evaluation of J			
r	ADOL-C	MAPLE	ADOL-C	MAPLE	G	J
1	0.0077	0.0010	0.0209	0.0029	33.93	76.11
2	0.0087	0.0013	0.0299	0.0049	87.54	188.57
3	0.0094	0.0016	0.0414	0.0082	190.55	403.80

Average Memory used in Kilobytes.

	MAPLE		ADOL-C	
r	G	J	G	J
1	1014	1305	72.9	109.3
2	1133	1681	94.6	120.4
3	1247	2386	98.9	161.9

7.4 High-Purity Absorption Column

The final example is also from chemical process control [18]. The process consists of a series of N trays. Each tray, if vapor dynamics are modeled, has an index two DAE model. The overall system is usually also index two. For the first and last trays there are 9 variables. For trays $i = 2, \ldots, N-1$ there are 10 variables. The example considered in the paper is index two and consists of 5 trays. It is an index two DAE in 48 variables. Only ADOL-C was used.

ADOL-C on the Absorption Column

	Evaluation CPU Time		Average Memory Used	
r	G	J	G	J
1	0.0416	0.0947	150.7	250.9
2	0.0466	0.1250	150.3	281.8

7.5 Comments on Computations

These additional computations support the previous observation in [10] that for a one time integration using the AD software is superior. However, if multiple runs are being done then on moderate sized problems the Jacobians from MAPLE are somewhat faster than the unoptimized AD Jacobians. With the shuttle we had AD taking about 3 times as long for G and 2 times as long for J with $r = 3$. A similar difference is noted for the robot arm at $r = 5$. With the chemical control problem, we see AD taking 6 times as long for G and 2 times as long for J. For problems where there are infrequent Jacobian updates the G time can actually dominate and the AD code would be around 6 times slower than the one using the MAPLE Jacobians.

These computations also illustrate the dramatic difference in memory use between MAPLE and ADOL-C. Theoretically the ADOL-C complexity grows quadratically with r. However, for $r < 10$, which includes many important classes of applications, the growth appears to be linear because memory traffic is proportional to r.

These examples show the necessity for AD software to have some sort of built in optimization if it is to be the method of choice for these types of integrators. We have done some experimentation with unoptimized MAPLE FORTRAN Jacobians. As reported in [8] for the shuttle problem, they took over 4 times as long to evaluate as the optimized MAPLE FORTRAN Jacobians. The G evaluation times between the optimized and unoptimized MAPLE FORTRAN codes differed by a factor of 6.

8 Comments on AD Software

We see that there are several distinguishing features of our application of AD.

1. We need both y_i and $\partial y_i/\partial u_i$.
2. We need higher derivatives than just first or second.
3. The equations are often of moderate size: 5–50.
4. G, J must be computed a large number of times: 100's – 1000's.
5. Discontinuities in the equations can occur in DAEs. Often, however, the equations stay the same for a large number of time steps.
6. If these methods are also to be reasonable to use on some problems for which other methods can be used then it is important that the evaluation be quick.
7. We have x' which are derivatives of other functions x.

It is clear that automatic differentiation is essential for our numerical methods if they are to be used on even moderately sized problems. In the context of our particular application, and given our computational experience as a user, we are led to make the following comments.

PROBLEM FORMULATION: It has been noted that problem formulation can greatly impact the speed of automatic differentiation algorithms. There is even more freedom in rewriting a DAE than for ODEs. For example, a term $x' = f(x)/g(x)$ can be written $g(x)x' = f(x)$. However, it is not clear what is the best way to formulate the equations. When is it worthwhile to first optimize the equations before applying AD? If it is worthwhile what type of optimization is the best? Should there be a special type of optimization with AD in mind? For example, which is better: $(x + y + z)^2$ or $x^2 + 2yx + 2zx + 2yz + y^2 + z^2$ or something else? This information needs to be

available to the user in clearly written guidelines which also describe the expected benefit to be gained.

AUTOMATIC OPTIMIZATION: For complex problems the need for the user to perform some optimization can be a disadvantage. This is particularly the case with an integrator where users of the integrator may not want to have to deal with learning the ins and outs of automatic differentiation. Software that could perform this optimization would be very useful. This optimization needs to be callable directly on the original equations. One clearly does not want to have to call it every time that AD is to be done.

FORTRAN: It would be nice to have a FORTRAN code that did what we needed. This may be coming with FORTAN 90.

INITIALIZATION TIME: On fairly small problems the MAPLE generated FORTRAN code is noticeably quicker than automatic differentiation. This is in part due to a "startup cost." AD starts from scratch each time it is called. In our problems the same G, J are being repeatedly evaluated on the same functions. Is there a way to save the graphical information to speed up future evaluations? Such an option might increase storage requirements for AD but there might be a size range where it would be advantageous. The ability to trade away storage for speed can be a welcome user option.

INTERFACE: Given that one has written an AODL-C driver, then the programming effort needed to use MAPLE or ADOL-C is essentially identical. However, the driver has to translate the automatic differentiation results into the actual results needed whereas MAPLE produces directly what is wanted. User interfaces which allow the potential user to be able to directly request the output in the form they want would be very useful.

ALTERNATIVE ALGORITHMS: In some cases it might be worthwhile to think of developing software in terms of AD concepts rather than in the usual terms. As noted, in our particular case the Jacobian J has a pattern to it. However, this pattern was not obvious in the usual variables. It was only while working with the ADOL-C output where the pattern is obvious did we find out about the pattern in J that can be used even with a symbolic approach. If algorithms could be recast directly in terms of automatic differentiation output, then the banded structure might be exploitable thus greatly speeding up the linear algebra. We have looked at this problem a little bit but have nothing positive to report. This could be a major project but it might pay big dividends on some problems.

VARIABLE DEPENDENCIES: Situations like ours where we have $F(x, v)$ and $v = x'$ arise frequently in ODEs and PDEs. The ability of the software to directly handle these types of dependencies would be helpful.

References

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, 1996.

[2] S. L. Campbell, *A computational method for general higher index nonlinear singular systems of differential equations*, IMACS Transactions on Scientific Computing, 1.2 (1989), pp. 555–560.

[3] ——, *Numerical methods for unstructured higher index DAEs*, Annals of Numerical Mathematics, 1 (1994), pp. 265–278.
[4] ——, *Least squares completions for nonlinear differential algebraic equations*, Numer. Math., 65 (1993), pp. 77–94.
[5] ——, *High index differential algebraic equations*, J. Mech. Struct. & Machines, 23 (1995), pp. 199–222.
[6] S. L. Campbell and C. W. Gear, *The index of general nonlinear DAEs*, Numerische Mathematik, 72 (1995), pp. 173-196.
[7] S. L. Campbell and E. Griepentrog *Solvability of general differential algebraic equations*, SIAM J. Sci. Comp., 16 (1995), pp. 257–270.
[8] S. L. Campbell and E. Moore, *Progress on a general numerical method for nonlinear higher index DAEs II*, Circuits, Systems, Signal Processing, 13 (1994), pp. 123–138.
[9] ——, *Constraint preserving integrators for general nonlinear higher index daes*, Numerische Mathematik, 69 (1995), pp. 383–399.
[10] S. L. Campbell, E. Moore, and Y. Zhong, *Utilization of automatic differentiation in control algorithms*, IEEE Trans. Automatic Control, 39 (1994), pp. 1047–1052.
[11] B. Christianson, *Automatic Hessians by reverse accumulation in Ada*, IMA J. Numerical Anal., 12 (1992), pp. 135-150.
[12] A. De Luca and A. Isidori, *Feedback linearization of invertible systems*, Colloq. Aut. & Robots, Duisburg, 1987.
[13] A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, 1991.
[14] A. Griewank, D. Juedes, and J. Srinivasan, *ADOL-C: a package for the automatic differentiation of algorithms written in C/C++*, Preprint MCS-P180-1190, Mathematics and Computer Science Division, Argonne National Laboratory, 1991.
[15] E. Hairer, C. Lubich, and M. Roche, *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Springer-Verlag, New York, 1989.
[16] A. Kumar and P. Daoutidis, *Feedback control of nonlinear differential-algebraic-equation systems*, AIChE Journal, 41 (1995), pp. 619–636.
[17] ——, *Feedback regularization and control of nonlinear differential-algebraic-equation systems*, Preprint, 1995.
[18] ——, *Control of nonlinear differential-algebraic-equation systems with disturbances*, IEC Research, to appear.
[19] G. Meyer, *Dynamic forms and their application to control*, NASA Technical Memorandum NASA TM-103936, 1992.

9 Appendix: Robot Arm

To give some idea of the complexity of the test problems, this appendix provides the equations for the second test example. This example is a slight simplification of the equations for the prescribed path control of a two-link, flexible joint, planar robotic arm [12]. The equations used here are from [7].

$$x_1' = x_4 \tag{21a}$$

$$x_2' = x_5 \tag{21b}$$

$$x_3' = x_6 \tag{21c}$$

$$\begin{aligned} x_4' &= 2c(x_3)(x_4+x_6)^2 + x_4^2 d(x_3) + (2x_3 - x_2)(a(x_3) + 2b(x_3)) \\ &\quad + a(x_3)u_1 - a(x_3)u_2 \end{aligned} \tag{21d}$$

$$\begin{aligned} x_5' &= -2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) + (2x_3 - x_2)(1 - 3a(x_3) - 2b(x_3)) \\ &\quad - a(x_3)u_1 + (a(x_3)+1)u_2 \end{aligned} \tag{21e}$$

$$x_6' = -2c(x_3)(x_4+x_6)^2 - x_4^2 d(x_3) + (2x_3 - x_2)(a(x_3) - 9b(x_3))$$

$$-2x_4^2 c(x_3) - (x_4+x_6)^2 d(x_3) - (a(x_3)+b(x_3))(u_1+u_2) \tag{21f}$$

$$0 = \cos x_1 + \cos(x_1+x_3) - p_1(t) \tag{21g}$$

$$0 = \sin x_1 + \sin(x_1+x_3) - p_2(t) \tag{21h}$$

where

$$p_1(t) = \cos(e^t - 1) + \cos(t-1) \tag{22a}$$

$$p_2(t) = \sin(1-e^t) + \sin(1-t) \tag{22b}$$

$$a(s) = \frac{2}{2-\cos^2 s}, \quad b(s) = \frac{\cos s}{2-\cos^2 s} \tag{22c}$$

$$c(s) = \frac{\sin s}{2-\cos^2 s}, \quad d(s) = \frac{\cos s \sin s}{2-\cos^2 s} \tag{22d}$$

$p(t)$ is the path. Equations (21) and (22) form a DAE in $\{x_1, \ldots, x_6, u_1, u_2\}$. By construction, part of the solution of (21) is $x_1 = 1 - e^t, x_3 = e^t - t$.