ELSEVIER

Journal of Computational and Applied Mathematics 125 (2000) 131–145

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.nl/locate/cam

Software and algorithms for sensitivity analysis of large-scale differential algebraic systems ☆

Shengtai Li, Linda Petzold *

Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

Received 22 June 1999; received in revised form 21 March 2000

Abstract

Sensitivity analysis for DAE systems is important in many engineering and scientific applications. The information contained in the sensitivity trajectories is useful for parameter estimation, optimization, model reduction and experimental design. In this paper we present algorithms and software for sensitivity analysis of large-scale DAE systems of index up to two. The new software provides for consistent initialization of the solutions and the sensitivities, interfaces seamlessly with automatic differentiation for the accurate evaluation of the sensitivity equations, and is capable via MPI of exploiting the natural parallelism of sensitivity analysis as well as providing an efficient solution in sequential computations.

1. Introduction

This paper is concerned with the solution and sensitivity analysis of initial value problems for differential–algebraic equation systems (DAEs) in the general form

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

where F, y, and y' are N-dimensional vectors. A number of powerful solvers have been written for the solution of DAEs, including RADAU5 [9], SPRINT [1], PSIDE [17], DASSL [3] and DASPK [4]. Many of these solvers are compared in [13], where it was found that DASSL/DASPK works very well compared with other methods and can solve a broader class of problems than the other codes tested. Several methods and codes have been designed in the last decade to compute sensitivities for DAEs [14,6]. In this paper we outline the algorithms and issues for sensitivity analysis of large-scale DAE systems, describe our new software DASPK3.0 for these problems and present some numerical

☆ This work was partially supported by DOE contract number DE-FG03-98ER25354, NSF grant CCR-9896198, NSF KDI grant ATM-9873133, NSF/DARPA grant DMS-9615858, and LLNL ISCR 99-006.

* Corresponding author.

0377-0427/00/$ - see front matter
PII: S0377-0427(00)00464-7

examples illustrating the effectiveness of this approach. The sensitivity algorithms presented here are applicable to most solvers. We have based our software on DASPK so as to make use of the excellent properties, particularly for large-scale systems, of this powerful and widely used code.

We begin by giving some background in Section 2 on the basic algorithms used in the DAE solver DASSL and its extension DASPK for large-scale DAE systems. In Section 3 we describe three algorithms for sensitivity analysis of DAEs using a direct approach. These are the staggered direct method [6], simultaneous corrector method [14] and staggered corrector method [8]. Accurate evaluation of the sensitivity residuals is an important problem; an adaptive increment finite difference method is presented, and methods for computing residuals via automatic differentiation are described. In Section 4 we present methods for determining consistent initial conditions for several classes of index-0, index-1 and index-2 DAEs and the associated sensitivities. The methods, which are new for index-2 DAEs, have the important property that they can be implemented with very little additional information required from the user. In Section 5, several issues which are critical for a robust and efficient implementation of sensitivity analysis are explored, along with their resolution in DASPK3.0. These include the error and convergence tests and formulation of the Krylov method for sensitivity analysis. The method used for parallelization in DASPK3.0 is described in Section 6. Finally, numerical experiments illustrating the effectiveness of this approach on both sequential and parallel computers are given in Section 7.

Further details on the implementation and use of DASPK3.0 are given in [11]. DASPK3.0 is available via "http://www.engineering.ucsb.edu/~cse".

2. Background – Algorithms in DASSL and DASPK

DASSL was developed by Petzold [3] and has become one of the most widely used software packages for DAEs. DASSL uses backward differentiation formula (BDF) methods [3] to solve a system of DAEs or ODEs. The methods are variable step size, variable order. The system of equations is written in implicit ODE or DAE form as in (1). Following discretization by the BDF methods, a nonlinear equation

$$F(t, y, \alpha y + \beta) = 0 \tag{2}$$

must be solved at each time step, where $\alpha = \alpha_0/h_n$ is a constant which changes whenever the step size or order changes, β is a vector which depends on the solution at past times, and t, y, α, β are evaluated at t_n. DASSL solves this equation by a modified version of Newton's method

$$y^{(m+1)} = y^{(m)} - c\left(\alpha\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}\right)^{-1} F(t, y^{(m)}, \alpha y^{(m)} + \beta), \tag{3}$$

where the linear system is solved via a dense or banded direct solver. The iteration matrix

$$A = \alpha\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}$$

is computed and factored and is then used for as many time steps as possible. The reader can refer to [3] for more implementation details.

DASPK2.0 was developed by Brown et al. [4] for the solution of large-scale systems of DAEs. It is particularly effective in the method-of-lines solution of time-dependent PDE systems in two and

three dimensions. In contrast to DASSL, which is limited in its linear algebra to dense and banded systems, DASPK2.0 is able to make use of the preconditioned GMRES iterative method [16] for solving the linear system at each Newton iteration.

When solving DAEs, the integration must be started with a consistent set of initial conditions y_0 and y_0'. This is a set of initial conditions which satisfy the algebraic constraints for the DAEs (and for higher-index DAEs, the hidden constraints). The initialization algorithms in DASPK3.0 are new, and will be discussed further in Section 4.

3. Sensitivity analysis of DAEs

3.1. Methods for sensitivity analysis

Several approaches have been developed to calculate sensitivity coefficients [6,14]. Here we summarize the direct methods for sensitivity analysis of ODEs and DAEs.

To illustrate the basic approach for sensitivity analysis, consider the general DAE system with parameters,

$$F(t, y, y', p) = 0, \quad y(0) = y_0, \tag{4}$$

where $y \in \mathbb{R}^{n_y}$, $p \in \mathbb{R}^{n_p}$. Here n_y is the number of time-dependent variables y as well as the dimension of the DAE system, and n_p is the number of parameters in the original DAE system. Sensitivity analysis entails finding the derivative of the solution y with respect to each parameter. This produces an additional $n_s = n_p \cdot n_y$ sensitivity equations which, together with the original system, yield

$$\begin{aligned} &F(t, y, y', p) = 0, \\ &\frac{\partial F}{\partial y} s_i + \frac{\partial F}{\partial y'} s_i' + \frac{\partial F}{\partial p} = 0, \quad i = 1, \ldots, n_p, \end{aligned} \tag{5}$$

where $s_i = \mathrm{d}y/\mathrm{d}p_i$. Defining

$$Y = \begin{bmatrix} y \\ s_1 \\ \vdots \\ s_{n_p} \end{bmatrix}, \quad \mathrm{F} = \begin{bmatrix} F(t, y, p) \\ \dfrac{\partial F}{\partial y} s_1 + \dfrac{\partial F}{\partial y'} s_1' + \dfrac{\partial F}{\partial p_1} \\ \vdots \\ \dfrac{\partial F}{\partial y} s_{n_p} + \dfrac{\partial F}{\partial y'} s_{n_p}' + \dfrac{\partial F}{\partial p_{n_p}} \end{bmatrix},$$

the combined system can be rewritten as

$$\mathrm{F}(t, Y, Y', p) = 0, \quad Y(0) = \begin{bmatrix} y_0 \\ \dfrac{\mathrm{d}y_0}{\mathrm{d}p_1} \\ \vdots \\ \dfrac{\mathrm{d}y_0}{\mathrm{d}p_{n_p}} \end{bmatrix}.$$

This system can be solved by the kth order BDF formula with step size h_{n+1} to yield a nonlinear system

$$G(Y_{n+1}) = \mathrm{F}\left(t_{n+1}, Y_{n+1}, Y'^{(0)}_{n+1} - \frac{\alpha_s}{h_{n+1}}(Y_{n+1} - Y^{(0)}_{n+1}), p\right) = 0, \tag{6}$$

where $Y^{(0)}_{n+1}$ and $Y'^{(0)}_{n+1}$ are predicted values for Y_{n+1} and Y'_{n+1}, which are obtained via polynomial extrapolation of past values [3]. Also, α_s is the fixed leading coefficient which is defined in [3]. Newton's method for the nonlinear system produces the iteration

$$Y^{(k+1)}_{n+1} = Y^{(k)}_{n+1} - \boldsymbol{J}^{-1} G(Y^{(k)}_{n+1}),$$

where

$$\boldsymbol{J} = \begin{bmatrix} J & & & & \\ J_1 & J & & & \\ J_2 & 0 & J & & \\ \vdots & \vdots & \vdots & \ddots & \\ J_{n_p} & 0 & \cdots & 0 & J \end{bmatrix} \tag{7}$$

and

$$J = \alpha \frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}, \quad J_i = \frac{\partial J}{\partial y} s_i + \frac{\partial J}{\partial p_i}$$

and $\alpha = \alpha_s / h_{n+1}$.

There are three well-established methods to solve the nonlinear system (6):

- Staggered direct method, described in [6].
- Simultaneous corrector method, described in [14].
- Staggered corrector method, described in [8].

Analysis and comparison of the performance of these three methods have been given in [8,12]. Because the relative efficiencies of the methods depend on the problem and on the number of parameters, all of them were made available as options in DASPK3.0. Here we describe briefly the three methods.

The staggered direct method first solves Eq. (6) for the state variables. After the Newton iteration for the state variables has converged, the sensitivity equations in (6) are updated with the most recent values of the state variables. Because Eq. (8) is linear with a matrix J for the sensitivity

equations, it is solved directly without Newton iteration. However, to solve the linear system in this way requires computation and factorization of the Jacobian matrix at each step and also extra storage for the matrix $\partial F/\partial y'$. Since the Jacobian is updated and factorized only when necessary in DASPK, the additional matrix updates and factorizations may make the staggered direct method unattractive compared to the other methods. However, if the cost of a function evaluation is more than the cost of factorization of the Jacobian matrix and the number of sensitivity parameters is very large (see [12]), the staggered direct method is more efficient. We have modified the implementation of [6] to make the staggered direct method more reliable for ill-conditioned problems.

The simultaneous corrector method solves (6) as one whole nonlinear system, where Newton iteration is used. The Jacobian matrix $\boldsymbol{J}$ in (7) is approximated by its block diagonal in the Newton iteration. Thus, this method allows the factored corrector matrix to be reused for multiple steps. It has been shown in [14] that the resulting iteration is two-step quadratically convergent for full Newton, and convergent for modified Newton iteration.

The staggered corrector method is similar to the staggered direct method. However, instead of solving the linear sensitivity system directly, a Newton iteration is used

$$s_i^{(k+1)} = s_i^{(k)} - J^{-1} G_{s_i}(s_i^{(k)}), \tag{8}$$

where G_{s_i} is the residual for the ith sensitivity and J is the factored Jacobian matrix which is used in the Newton iteration for the state variables. Like the simultaneous corrector method, this method does not require the factorization of the Jacobian matrix at each step. One of the advantages of the staggered corrector method is that we do not need to evaluate the sensitivity equations during the iteration of solving for the state variables. This can reduce the computation time if the state variables require more iterations than the sensitivity variables. After solving for the state variables in the corrector iteration, only the diagonal part of $\boldsymbol{J}$ in (7) is left. We can expect the convergence of the Newton iteration will be improved over that of using an approximate iteration matrix in the simultaneous corrector method. This has been observed in our numerical experiments.

3.2. Methods for evaluating sensitivity residuals

Several approaches have been developed to calculate the sensitivity residuals that may be used with either the staggered corrector or the simultaneous corrector methods. Maly and Petzold [14] used a directional derivative finite difference approximation. For example, the ith sensitivity equation may be approximated as

$$\frac{F(t, y + \delta_i s_i, y' + \delta_i s_i', p + \delta_i e_i) - F(t, y, y', p)}{\delta_i} = 0, \tag{9}$$

where δ_i is a small scalar quantity, and e_i is the ith unit vector. Proper selection of the scalar δ_i is crucial to maintaining acceptable round-off and truncation error levels [14]. If $F(t, y, y', p)$ is already available from the state equations, which is the case in the Newton iteration of DASPK, (9) needs only one function evaluation for each sensitivity. The main drawback of this approach is that it may be inaccurate for badly scaled problems.

The selection of the increment δ_i for Eq. (9) in DASPK3.0 is an improvement over the algorithms of [14] which was suggested by Hindmarsh [10]. The increment is given by

$$\delta_i = \Delta \max(|p_i|, 1/\|u_i\|_2), \tag{10}$$

where Δ is a scale factor

$$u_i = \left(WT^{in_y+j}/WT^j\colon j = 1,\ldots,n_y\right)$$

and WT is a vector of weights determined by the relative and absolute user error tolerances and the solution y,

$$WT^j = \mathrm{RTOL}_j \cdot |y_j| + \mathrm{ATOL}_j.$$

Alternatively, the sensitivity residuals can be evaluated analytically by an automatic differentiation tool such as ADIFOR [2] or other automatic differentiation (AD) methods. We recommend using AD to evaluate the sensitivity equations. Even for some well-scaled problems, the ADIFOR-generated routine has better performance in terms of efficiency and accuracy than the finite difference approximation.

All of the ADIFOR-generated routines require the support of the ADIFOR library [2]. For further information, see http://www-unix.mcs.anl.gov/autodiff/-ADIFOR/. DASPK3.0 can be used without the ADIFOR library, for problems where automatic differentiation is not needed.

4. Consistent initial condition calculation for solution and sensitivity analysis

4.1. Consistent initial conditions for index-one problems

The basic initialization technique in DASPK3.0 for index-1 problems is an extension of the method proposed in [5]. It is applicable to two classes of index-1 initialization problems. Initialization problem I is posed for systems of the form

$$f(t,u,v,u') = 0,$$

$$g(t,u,v) = 0, \tag{11}$$

where $u, f \in \mathbb{R}^{N_d}$ and $v, g \in \mathbb{R}^{N_a}$, with the matrices $f_{u'} = \partial f/\partial u'$, $g_v = \partial g/\partial v$ square and nonsingular. The problem is to find the initial value v_0 of v when the initial value u_0 for u is specified. Hence it is required for the user to specify which variables are algebraic and which are differential.

In initialization problem II, which is applicable to the general index-1 system (1), the initial derivatives are specified but all of the dependent variables are unknown. That is, we must solve for y_0 given y_0'. For example, beginning the DAE solution at a steady state corresponds to specifying $y_0' = 0$.

Both of these initial condition problems are solved with the help of mechanisms already in place for the solution of the DAE system itself, rather than requiring the user to perform a special computation. It is also possible in DASPK3.0 to specify some of the solution variables and derivatives at the initial time, and solve for the rest. This will be described in more detail in the next subsection.

The sensitivity problem for (11) is given by

$$f(t,u,v,u',p) = 0,$$

$$g(t,u,v,p) = 0,$$

$$\frac{\partial f}{\partial u}s_u + \frac{\partial f}{\partial v}s_v + \frac{\partial f}{\partial u'}s_{u'} + \frac{\partial f}{\partial p} = 0,$$

$$\frac{\partial g}{\partial u}s_u + \frac{\partial g}{\partial v}s_v + \frac{\partial g}{\partial p} = 0. \tag{12}$$

The algebraic variables in Eq. (11) generate algebraic sensitivity variables in Eq. (12). Eq. (12) also has the same index as (11).

DASPK3.0 uses a staggered approach to compute the consistent initial conditions for the sensitivity variables. First the consistent initial conditions are computed for the state variables, and then for the sensitivity variables. It is easy to see by differentiating the initial conditions that the sensitivity variables fall into the same class of initialization problems as the state variables.

4.2. Consistent initial conditions for index-two problems

With partial error control (excluding the algebraic variables from the error control), DASPK3.0 can solve Hessenberg index-2 problems with given consistent initial conditions. However, consistent initial conditions may not be readily available in many cases. Taking the Hessenberg index-2 problem as an example,

$$\begin{aligned} u' &= f(t,u,v), \\ 0 &= g(u), \end{aligned} \tag{13}$$

the objective for index-2 initialization is to compute a new triple $(\hat{u}_0', \hat{u}_0, \hat{v}_0)$ that satisfies the constraints and consistent initial conditions. The problem is under-determined. Following the idea of [5] for index-1 problems, we solve the consistent initialization problem with the help of mechanisms already in place for the DAE solution itself. We search for the consistent initial conditions in the direction given by the differential equations. This method has a potential advantage that the hidden constraints derived from the equations may also be satisfied. To do that, we should increment the derivative u' by $(1/h)\delta u$ if the solution u is incremented by δu. Consider a general DAE system

$$F(t,u,u',v) = 0. \tag{14}$$

After introducing two new variables δu and δv and an artificial time step h, we transform Eq. (14) into

$$F\left(t, u_0 + \delta u, u_0' + \frac{1}{h}\,\delta u, v_0 + \delta v\right) = 0. \tag{15}$$

Then δu and δv in (15) are computed by Newton iteration with initial values of zero. The iteration matrix is

$$J = \left(\frac{1}{h}F_{u'} + F_u, F_v\right), \tag{16}$$

where h is chosen to be the initial step size that satisfies the error tolerance for a zeroth-order method.

It is easy to fix some of the differential variables in (15). However, fixing a differential variable u does not imply fixing the derivative u' in our algorithm, and vice versa. For example, if we fix the first element u_{01} of the vector u in (15), the equation becomes

$$F\left(t, u_{01}, u_{0r} + \delta u_r, u'_0 + \frac{1}{h}\delta u, v_0 + \delta v\right) = 0,$$

where u_{0r} is the rest of u (excluding u_{01}), and δu_r is the rest of δu (excluding δu_1). In the algorithm of [5] for initialization problem I, all of the differential variables are fixed and Eq. (15) becomes

$$F\left(t, u_0, u'_0 + \frac{1}{h}\delta u, v_0 + \delta v\right) = 0. \tag{17}$$

The initialization problem II for Eq. (11) can also be cast into (15) by fixing all of the derivatives u', which yields

$$F(t, u_0 + \delta u, u'_0, v_0 + \delta v) = 0.$$

As in [5], the implementation is designed so that forming the Jacobian for the initialization and solving the nonlinear system requires no information from the user beyond what is already needed for the time stepping.

If the constraint $g(u) = 0$ in (13) is satisfied, all of the differential variables can be fixed and Eq. (17) becomes

$$\begin{aligned} &\delta u' + u'_0 = f(t, u_0, v_0 + \delta v), \\ &0 = g(u_0), \end{aligned} \tag{18}$$

where $\delta u' = (1/h)\delta u$. Since $\delta u'$ and δv are not related to $g(u_0)$, the iteration matrix for (18) is singular, which means the solution is not unique. However, we can replace the constraint equation in (18) with $g_u u' = 0$, which yields

$$\begin{aligned} &\delta u' + u'_0 = f(t, u_0, v_0 + \delta v), \\ &0 = g_u(u_0)(\delta u' + u'_0). \end{aligned} \tag{19}$$

It is easy to evaluate the first equation in (19), however the second equation is not available to DASPK3.0. To evaluate it requires the user to specify which equations are algebraic. We can avoid evaluating $g_u u' = 0$ if $f(t, u, v)$ is linear in v. Note that if (19) is linear with respect to $u' = \delta u' + u'_0$ and $v = v_0 + \delta v$, it has a unique solution for u' and v. The u' and v can be solved via only one iteration for a linear system, independent of the initial values. If we set $u'_0 = 0$ and $\delta u' = 0$ in our first guess, the value of the second equation in (19) is zero, which is also the result of $g(u_0)$. Therefore, the residual evaluations can be used without modification. If $f(t, u, v)$ is nonlinear with respect to v, then it might take more than one iteration to solve for u' and v. Since $g_u u'$ might not be zero during the intermediate iterations, $g_u u'$ must be evaluated in addition to the residual evaluations. If $f(t, u, v)$ is nonlinear with respect to v, the user can either evaluate the $g_u u'$ in the residual routine or specify which equations are algebraic and DASPK3.0 will compute $g_u u'$ automatically, via finite difference approximation or automatic differentiation.

In our implementation, a linesearch backtracking algorithm [5] has been used to improve the robustness of the Newton algorithm for the initial condition calculation.

5. Implementation issues

5.1. Error and convergence tests

In DASPK3.0, the norm is evaluated separately for the state variables and for the sensitivities with respect to each parameter. For the error and convergence tests, we choose the largest one among all the norms. We have found from experience that this leads to the most robust and efficient implementation. It is possible to exclude sensitivity variables from the error test, but not from the Newton iteration convergence test.

5.2. Staggered direct method

In the staggered direct method as described in Caracotsios and Stewart [6], system (6) is transformed into

$$Js_{i_{(n+1)}} = \left(-\frac{\partial F}{\partial y'_{n+1}}\beta - \frac{\partial F}{\partial p_i} \right), \tag{20}$$

where $\beta_i = s'^{(0)}_{i_{(n+1)}} - \alpha s^{(0)}_{i_{(n+1)}}$. To solve a linear system in this way requires extra storage for the matrix $\partial F/\partial y'_{n+1}$. Moreover, this implementation often fails when the matrix J is ill-conditioned. This is because the right-hand side of Eq. (20) can be very large and can introduce large round-off errors when J is ill-conditioned [12].

In DASPK3.0, the following linear system is solved for the sensitivities:

$$J\delta = Js^{(0)}_{i_{(n+1)}} + \frac{\partial F}{\partial y'_{n+1}}\beta\frac{\partial F}{\partial p_i}, \tag{21}$$

where $\delta = s^{(0)}_{i_{(n+1)}} - s_{i_{(n+1)}}$. The right-hand side of (21) is easy to obtain in DASPK3.0 by evaluation of the sensitivity equations. It does not require any extra storage or special handling. What is important is that it works well for ill-conditioned problems. This is because the right-hand side of Eq. (21) is usually much smaller than that of Eq. (20) for a successful step (which means the predicted value $s^{(0)}$ is close enough).

5.3. Krylov method

Since the sensitivity equations are linear with respect to the sensitivity variables, Newton iteration is not necessary for the staggered method. Therefore the staggered corrector method and staggered direct method are the same for the preconditioned Krylov iteration. The matrix–vector product Jv_{s_i} is evaluated directly via directional derivative finite difference approximation

$$\begin{aligned} Jv_{s_i} &= (\alpha F_{y'} + F_y)v_{s_i} \\ &\approx \frac{F(t, y + \sigma v_{s_i}, \alpha(y + \sigma v_{s_i}) + \beta, p) - F(t, y, \alpha y + \beta, p)}{\sigma}, \end{aligned} \tag{22}$$

where $F(t, y', y, p)$ are the state equations. The function evaluations in (22) involve only the state equations. Because there is no coupling between different sensitivity variables, the linear iteration for

each sensitivity equation can be done separately, which allows us to split the large linear system into several small ones and reduce the length of each orthonormal basis to the number of state variables. For the simultaneous corrector method, we approximate the Newton–Krylov iteration matrix by its block diagonal as for the direct method. Then (22) can be used to calculate the matrix–vector product.

One might consider replacing all finite differencing with ADIFOR-generated routines. However, this does not turn out to be a good idea for the Krylov iteration. There is a trade-off when we consider the efficiency and accuracy of the computations. The ADIFOR-generated routine not only computes the derivatives but also the original functions. To compute one matrix–vector product in an ADIFOR-generated routine requires at least one evaluation of the original function and possibly more than one evaluation of the derivatives. But the matrix–vector product approximated by first-order finite difference requires only one evaluation of the original functions. Since the finite difference approximation in the matrix–vector product for the Krylov iteration has the scaling incorporated into its implementation, in practice it has been quite robust.

6. Parallel implementation of DASPK for sensitivity analysis

Several parallel implementations for sensitivity analysis of DAEs via message-passing interface (MPI) [7] have been compared in [15]. In this section, we describe the parallelization in DASPK3.0. Although all the tests in [15] are for DASSL with the direct method, the comparative results are similar for DASPK3.0 with both the direct method and the Krylov method with the new implementation described in Section 5.3. We have found that the distributed parameter only (DPO) approach of [15] is also the fastest for DASPK3.0.

Our implementation distributes the sensitivity parameters inside the DASPK code so as to reduce the burden on the user. To balance the workload between processors, we allocate the parameters randomly to each processor: if we have NP processors and NPAR parameters, $N = \mathrm{NPAR}/\mathrm{NP}$, we distribute parameter numbers

$$\begin{aligned} &j, \ldots, j + i * \mathrm{NP}, \ldots, j + N * \mathrm{NP} \quad \text{if } j \leqslant \mathrm{mod}(\mathrm{NPAR},\ \mathrm{NP}), \\ &j, \ldots, j + i * \mathrm{NP}, \ldots, j + (N-1) * \mathrm{NP} \quad \text{if } j > \mathrm{mod}(\mathrm{NPAR},\ \mathrm{NP}), \end{aligned} \tag{23}$$

to the jth processor. Each processor computes the state variables locally, and the Jacobian matrix is also computed and factorized locally when needed. To minimize the storage and memory requirements in each processor, we assume that each processor has distributed memory, i.e., each processor has a local value of the same variable. Therefore, the work space in each processor can be reduced to approximately 1/NP of the total work space. Since the sensitivities are independent of each other, each processor can work independently without communicating with the others.

We have attempted to develop software for which both parallel and serial computation can run efficiently. We enforce the same step size control for all the processors in the parallel implementation. The communication overhead is very small. In each time step, each processor may be using different orders of the BDF formulae. Since this implementation requires an MPI-related routine and the support of the MPI library, which may not be accessible by users doing serial computation, we

provide a dummy routine which can be linked without involving the MPI library, for use with serial computation

7. Numerical experiments

In this section we describe several numerical experiments. All tests were run on an SGI O2 workstation. The following quantities are used to compare different methods:

METH	Integration method
NSTP	Number of time steps used
NRES	Number of calls to residual subroutine
NSE	Number of sensitivity evaluations
NJAC	Number of Jacobian evaluation
NNI	Number of nonlinear iterations
NLI	Number of linear iterations (only for Krylov method)
CPU	The total cpu time taken to solve the problem

The integration methods we use include the direct method (D) and Krylov method (K). The integration methods for the sensitivity equations include the staggered corrector method (ST), the staggered direct method (SD) and simultaneous corrector method (SI). Therefore we use STD to represent the staggered corrector direct method, STK to represent the staggered corrector Krylov method, SID to represent the simultaneous corrector direct method, SIK to represent the simultaneous corrector Krylov method, and SDK to represent the staggered direct Krylov method.

The first example models a multi-species food web [5], in which mutual competition and/or predator–prey relationships in the spatial domain are simulated. Specifically, the model equations for the concentration vector $c = (c^1, c^2)^{\mathrm{T}}$ are

$$c_t^1 = f_1(x, y, t, c) + (c_{xx}^1 + c_{yy}^1),$$

$$0 = f_2(x, y, t, c) + 0.05(c_{xx}^2 + c_{yy}^2)$$

with

$$f_i(x, y, t, c) = c^i \left(b_i + \sum_{j=1}^{2} a_{ij} c^j \right).$$

The coefficients a_{ij}, b_i are

$$a_{11} = a_{22} = -1, \quad a_{12} = -0.5 \cdot 10^{-6}, \quad a_{21} = 10^4,$$

$$b_1 = -b_2 = 1 + \alpha x y + \beta \sin(4\pi x) \sin(4\pi y).$$

The domain is the unit square $0 \leqslant x, y \leqslant 1$ and $0 \leqslant t \leqslant 10$. The boundary conditions are of Neumann type with normal derivative equal to 0. The PDEs are discretized by central differencing on an $M \times M$ mesh, for $M = 20$. Therefore the resulting DAE system has size $\mathrm{NEQ} = 2M^2 = 800$. The tolerances used were $\mathrm{RTOL} = \mathrm{ATOL} = 10^{-5}$.

Table 1
Results for multi-species food web. The upper part is for the ADIFOR option with error control including the algebraic variables. The middle part is for the ADIFOR option with error control excluding the algebraic variables. The bottom part is for the finite difference option with error control excluding the algebraic variables

METH	NSTP	NRES	NSE	NJAC	NNI	NLI	NLIS	NETF	CPU
STD	312	770	389	45	381	0	0	4	30.84
SID	335	508	508	42	508	0	0	3	36.56
STK	341	2712	353	36	406	732	0	1	22.98
SIK	505	4262	617	47	617	1532	0	9	39.12
STD	128	377	190	42	205	0	0	0	17.90
SID	128	228	228	40	228	0	0	0	18.91
STK	133	1456	147	38	165	329	425	0	11.36
SIK	131	1888	202	38	202	332	697	0	15.47
SDK	133	1442	133	38	165	329	425	0	11.03
STD	128	3589	190	42	187	0	0	0	24.85
SID	128	3240	228	40	228	0	0	0	26.11
STK	133	1442	147	38	165	329	425	0	10.36
SIK	131	1818	201	38	201	332	700	0	14.37
SDK	133	1442	133	38	165	329	425	0	10.12

For sensitivity analysis, α and β were taken as the sensitivity parameters with initial values $\alpha = 50$ and $\beta = 100$. The initial conditions were taken as

$$c^1 = 10 + (16x(1-x)y(1-y))^2,$$
$$c^2 = 10^5,$$

which does not satisfy the constraint equations. The initial conditions for the sensitivity variables were taken as zero, which are not consistent either. We solved this problem with both the direct and Krylov methods. For the Krylov methods, we used the block-grouping preconditioner (which is included in the package DASPK2.0 [5]). To eliminate the effect of finite differencing when comparing different methods, we used the ADIFOR option in DASPK3.0 to generate the Jacobian matrix (only for the direct method) and sensitivity equations. Without initialization, the integration failed because of too many convergence test failures. The consistent initial conditions were computed quickly with both the direct and Krylov methods. Table 1 shows the results of the staggered corrector method and the simultaneous corrector method. Full error control (including the sensitivity variables) was used. Although there were no convergence test failures for this problem, the staggered corrector method (ST) performed better than the simultaneous corrector method (SI).

The finite differencing options for the sensitivity equations were also tested. We used the central difference and $\Delta = 10^{-3}$ (default value). The results are shown in Table 1.

The next example is the heat equation,

$$\frac{\partial u}{\partial t} = p_1 u_{xx} + p_2 u_{yy},$$

Table 2
Results for heat equation with ADIFOR evaluation. The upper half is for partial error control (excluding the sensitivity variables). The bottom half is for full error control

METH	NSTP	NRES	NSE	NJAC	NNI	NLI	NETF	CPU
STD	64	160	65	22	95	0	3	36.20
SID	64	97	97	22	97	0	3	46.35
STK	71	1527	72	18	100	149	1	25.67
SIK	71	1572	102	18	102	184	1	29.40
STD	92	220	103	23	123	0	2	53.58
SID	93	130	130	25	130	0	3	63.63
STK	106	1823	114	24	141	182	2	35.68
SIK	116	1776	155	24	155	213	2	39.06

Table 3
Results for heat equation with finite difference approximation for sensitivities and full error-control

METH	NSTP	NRES	NSE	NJAC	NNI	NLI	NETF	CPU
STD	92	2118	103	23	117	0	2	64.07
SID	93	2175	130	25	130	0	3	75.76
STK	107	3917	114	24	143	187	2	38.39
SIK	116	3695	157	24	157	207	2	44.24

where $p_1 = p_2 = 1.0$, posed on the 2-D unit square with zero Dirichlet boundary conditions. An $M+2$ by $M+2$ mesh is set on the square, with uniform spacing $1/(M+1)$. The spatial derivatives are represented by standard central finite difference approximations. At each interior point of the mesh, the discretized PDE becomes an ODE for the discrete value of u. At each point on the boundary, we pose the equation $u = 0$. The discrete values of u form a vector U, ordered first by x, then by y. The result is a DAE system $G(t, U, U') = 0$ of size $(M+2) \times (M+2)$. Initial conditions are posed as

$$u(t = 0) = 16x(1 - x)y(1 - y).$$

The problem was solved by DASPK3.0 on the time interval $[0, 10.24]$ with $M = 40$. To compute the sensitivities, we took 10 sensitivity parameters; p_1 and p_2 were two of them. The other eight were chosen from the initial conditions. The error tolerances for DASPK are RTOL = ATOL = 1.0D − 4. For the direct method, we used ADIFOR with SparsLinC to generate the Jacobian. For the Krylov method, we used the incomplete LU (ILU) preconditioner, which is part of the DASPK package. The Jacobian for the ILU preconditioner was evaluated by ADIFOR with SparsLinC. The sensitivity residuals were evaluated by ADIFOR with the seed matrix option. Table 2 gives the results of the staggered corrector and simultaneous corrector methods.

Because this problem is linear and well scaled, finite-differencing in the Jacobian and/or sensitivity equation evaluation gets a good result. Table 3 shows the results when central differencing is used for evaluation of the sensitivity equations. The default perturbation factor (10^{-3}) is used in evaluating

Table 4
Results for heat equation with finite difference approximation and partial error-control. MPI was used in all the parallel computations. The same step size control was enforced on all the processors

METH	NPROC	NSTP	NRES	NJAC	NNI	NLI	CPU
	1	64	1810	19	91	0	35.33
Direct	2						19.64
	4						12.50
	8						8.78
	1	71	3410	18	100	181	24.59
Krylov	2	71	1935	18	100	159	12.94
	4	71	1109	18	100	160	7.43
	8	71	696	18	100	156	4.75

the sensitivity equations. The Jacobian is also evaluated by finite-differencing. Only the data for full error-control are listed.

We tested DASPK3.0 on a cluster of DEC alpha machines at Los Alamos National Laboratory. Each processor is 533 MHz with 64 MB memory. The heat equation with 24 sensitivity parameters was used as the test problem. The staggered corrector method was used. The synchronization to achieve the same step size on each processor does not introduce much overhead to the computation, as shown in Table 4.

The next example models a single pendulum

$$y_1' = y_3,$$

$$y_2' = y_4,$$

$$y_3' = -y_1 y_5,$$

$$y_4' = -y_2 y_5 - g,$$

$$0 = y_1 y_3 + y_2 y_4,$$

where g=1.0. This is an index-two problem. The initial conditions are y_1=0.5, y_2=$-\sqrt{p^2 - y_1^2}$, y_3= 10.0, $y_4 = 10.0$, and $y_5 = 0.0$. The sensitivity parameter is p, which has initial value $p = 1.0$. The initial conditions for the sensitivity variables are $(0.0, -1.1547, 0.0, 0.0, 0.0)$. All of the derivatives were set to 0 initially. The tolerance for DASPK was taken as $\mathrm{RTOL} = \mathrm{ATOL} = 10^{-6}$. Because

$$g(y) = y_1 y_3 + y_2 y_4 = -3.660254 \neq 0,$$

the consistent initial conditions were first computed via the initialization algorithm for index-2 problems. During the initial condition computation, we monitored three constraints,

$$g_1 = y_1^2 + y_2^2 - p,$$

$$g_2 = y_1 y_3 + y_2 y_4,$$

$$g_3 = y_3^2 + y_4^2 - (y_1^2 + y_2^2) y_5 - y_2.$$

Table 5
Results for consistent initial conditions for pendulum problem

Fixed	y_1	y_2	y_3	y_4	g_1	g_2	g_3
No	0.512	−0.859	11.777	7.016	−1.88e − 4	0.0	3.77e − 15
y_1, y_2	0.5	−0.866	11.83	6.83	−1.1e − 16	0.0	9.28e − 13

Initially, we have

$$g_1 = 0, \quad g_2 = -3.66, \quad g_3 = 200.866.$$

We also tried to fix y_1, y_2 during the experiments on the initial condition computation. The results are shown in Table 5. Note that if y_1 and y_2 are not fixed, g_1 may be violated.

References

[1] M. Berzins, P.M. Dew, R.M. Furzeland, Developing software for time-dependent problems using the method of lines and differential algebraic integrators, Appl. Numer. Math. 5 (1989) 375–397.
[2] C. Bischof, A. Carle, G. Corliss, A. Griewank, P. Hovland, ADIFOR-Generating derivative codes from Fortran programs, Sci. Programming 1(1) (1992) 11–29.
[3] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, Elsevier, New York, 1989, second edition, SIAM, Philadelphia, PA, 1996.
[4] P.N. Brown, A.C. Hindmarsh, L.R. Petzold, Using Krylov methods in the solution of large-scale differential-algebraic systems, SIAM J. Sci. Comput. 15 (1994) 1467–1488.
[5] P.N. Brown, A.C. Hindmarsh, L.R. Petzold, Consistent initial condition calculation for differential-algebraic systems, SIAM J. Sci. Comput. 19 (1998) 1495–1512.
[6] M. Caracotsios, W.E. Stewart, Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations, Comput. Chem. Eng. 9 (4) (1985) 359–365.
[7] N. Doss, W. Gropp, E. Luck, A. Skjellum, A model implementation of MPI, Technical Report, Argonne National Laboratory, 1993.
[8] W.F. Feehery, J.E. Tolsma, P.I. Barton, Efficient sensitivity analysis of large-scale differential-algebraic systems, Appl. Numer. Math. 25 (1997) 41–54.
[9] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, Springer, New York, 1991.
[10] A.C. Hindmarsh, personal communication.
[11] S. Li, L.R. Petzold, Design of new DASPK for sensitivity analysis, Technical Report, Department of Computer Science, University of California Santa Barbara, 1999.
[12] S. Li, L.R. Petzold, W. Zhu, Sensitivity analysis of differential-algebraic equations: a comparison of methods on a special problem, Appl. Numer. Math. 32 (2000) 161–174.
[13] W.M. Lioen, J.B. De Swart, Test set for initial value problem solvers, CWI Report MAS-R9832, 1998.
[14] T. Maly, L.R. Petzold, Numerical methods and software for sensitivity analysis of differential-algebraic systems, Appl. Numer. Math. 20 (1996) 57–79.
[15] L.R. Petzold, W. Zhu, Parallel sensitivity analysis for DAEs with many parameters, Concurrency: Practice Experience 11 (1999) 571–585.
[16] Y. Saad, M.H. Schulz, GMRES: A general minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 7 (1986) 856–869.
[17] J.B. De Swart, W.M. Lioen, W.A. van der Veen, Parallel software for implicit differential equations (PSIDE), http://www.cwi.nl/cwi/projects/PSIDE.