ELSEVIER

Journal of Computational and Applied Mathematics 149 (2002) 171–191

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software ☆

Yang Cao, Shengtai Li, Linda Petzold *

Department of Computer Science, University of California, Santa Barbara, CA, Engr III, Room 124, 93106-5070, USA

Received 6 May 2001; received in revised form 10 February 2002

Abstract

An efficient numerical method for sensitivity computation of large-scale differential-algebraic systems is developed based on the adjoint method. Issues that are critical for the implementation are addressed. Complexity analysis and numerical results demonstrate that the adjoint sensitivity method is advantageous over the forward sensitivity method for applications with a large number of sensitivity parameters and few objective functions.

Keywords: Sensitivity analysis; Adjoint method; Differential-algebraic equations

1. Introduction

Recent work on methods and software for sensitivity analysis of differential-algebraic equation DAE systems [7,9,13–16] has demonstrated that forward sensitivities can be computed reliably and efficiently via automatic differentiation [2] in combination with DAE solution techniques designed to exploit the structure of the sensitivity system. The DASPK3.0 [15,13] software package was developed for forward sensitivity analysis of DAE systems with index up to two, and has been used in sensitivity analysis and design optimization of several large-scale engineering problems [11,18]. DASPK3.0 is an extension of the DASPK software [1,3,4] developed by Brown et al. for the solution

☆ This work was supported by grants: EPRI WO-8333-06, NSF CCR 98-96198, NSF/ARPA PC 239415, NSF/KDI ATM-9873133 and LLNL ISCR 00-15.
* Corresponding author. Tel.: +1-805-893-5362; fax: +1-805-893-5435.
E-mail address: petzold@engineering.ucsb.edu (L. Petzold).

0377-0427/02/$ - see front matter
PII: S0377-0427(02)00528-9

of large-scale DAE systems. For a DAE depending on parameters,

$$F(x, \dot{x}, t, p) = 0,$$
$$x(0) = x_0(p), \tag{1}$$

these problems take the form: find $\mathrm{d}x/\mathrm{d}p_j$ at time T, for $j = 1, \ldots, n_p$. Their solution requires the simultaneous solution of the original DAE system with the n_p sensitivity systems obtained by differentiating the original DAE with respect to each parameter in turn. For large systems this may look like a lot of work but it can be done efficiently, if n_p is relatively small, by exploiting the fact that the sensitivity systems are linear and all share the same Jacobian matrices with the original system.

Some problems require the sensitivities with respect to a large number of parameters. For these problems, particularly if the number of state variables is also large, the forward sensitivity approach is intractable. These problems can often be handled more efficiently by the adjoint method [8]. In this approach, we are interested in calculating the sensitivity of an objective function

$$G(x, p) = \int_0^T g(x, t, p)\,\mathrm{d}t \tag{2}$$

or alternatively the sensitivity of a function $g(x, T, p)$ defined only at time T. The function g must be smooth enough that g_p and g_x exist and are bounded. While forward sensitivity analysis is best suited to the situation of finding the sensitivities of a potentially large number of solution variables with respect to a small number of parameters, reverse (adjoint) sensitivity analysis is best suited to the complementary situation of finding the sensitivity of a scalar (or small-dimensional) function of the solution with respect to a large number of parameters.

In [6] we derived the adjoint sensitivity system for DAEs of index up to two (Hessenberg) and investigated some of its fundamental properties. In this paper, we address some of the issues for the numerical solution and demonstrate the effectiveness of the adjoint method via our implementation of the adjoint DAE solver DASPKADJOINT.

The outline of this paper is as follows. In Section 2 we outline the adjoint sensitivity method for DAEs and summarize some of the relevant results from [6]. In Section 3 we describe how the adjoint DAE may be evaluated accurately and efficiently by an automatic differentiation tool. In Section 4 we study the consistent initialization of the adjoint DAE. Section 5 outlines some important considerations for implementation of the adjoint sensitivity method for DAEs and how they are addressed in our software, DASPKADJOINT. Finally, the algorithms and software are tested for several examples in Section 6. The numerical results show that the adjoint sensitivity method is advantageous over the forward sensitivity method for applications with a large number of sensitivity parameters and few derived functions.

2. The adjoint DAE system and sensitivity calculation

In this section we define the adjoint sensitivity system for DAEs and summarize some of the relevant results concerning initial values, stability and numerical stability from [6].

The adjoint system for the DAE

$$F(t, x, \dot{x}, p) = 0$$

with respect to the derived function $G(x, p)$ (2) is given by

$$(\lambda^* F_{\dot{x}})' - \lambda^* F_x = -g_x, \tag{3}$$

where * denotes the transpose operator and prime denotes the total derivative with respect to t.

The adjoint system is solved backwards in time. For index-0 and index-1 DAE systems, the initial conditions for (3) are taken to be $\lambda^* F_{\dot{x}}|_{t=T} = 0$, and the sensitivities of $G(x, p)$ with respect to the parameters p are given by

$$\frac{\mathrm{d}G}{\mathrm{d}p} = \int_0^T (g_p - \lambda^* F_p)\,\mathrm{d}t + (\lambda^* F_{\dot{x}})|_{t=0}(x_0)_p. \tag{4}$$

For Hessenberg index-2 DAE systems, the initial conditions are more complicated, and will be described in detail along with an algorithm for their computation in Section 4. For index-2 DAE systems, if the index-2 constraints depend on p explicitly, an additional term must be added to the sensitivity (4) [6].

For a scalar derived function $g(x, T, p)$, the corresponding adjoint DAE system is given by

$$(\lambda_T^* F_{\dot{x}})' - \lambda_T^* F_x = 0, \tag{5}$$

where λ_T denotes $\partial\lambda/\partial T$. For index-0 and index-1 DAE systems, the initial conditions $\lambda_T(T)$ for (5) satisfy $(\lambda_T^* F_{\dot{x}})|_{t=T} = [g_x - \lambda^* F_x]|_{t=T}$. We note that the initial condition $\lambda_T(T)$ is derived in such a way that the computation of $\lambda(t)$ can be avoided. This is the case also for index-2 DAE systems. The full algorithm for consistent initialization of the adjoint DAE system will be described in Section 4. The sensitivities of $g(x, T, p)$ with respect to the parameters p are given for index-0 and index-1 DAE systems by

$$\frac{\mathrm{d}g}{\mathrm{d}p} = (g_p - \lambda^* F_p)|_{t=T} - \int_0^T (\lambda_T^* F_p) + (\lambda_T^* F_{\dot{x}})|_{t=0}(x_0)_p. \tag{6}$$

Note that the values of both λ at $t = T$ and λ_T at $t = 0$ are required in (6). If $F_p \neq 0$, the transient value of λ_T is also needed. For an index-2 system, if the index-2 constraints depend on p explicitly, an additional term must be added to the sensitivity (6).

We focus on the adjoint system with respect to the scalar objective function $g(x, T, p)$ throughout this paper. If the objective function is of the integral form $G(x, p)$ (2), it can be computed easily by adding a *quadrature variable*, which is equal to the value of the objective function, to the original DAE. For example, if the number of variables in the original DAEs is N, we append a variable x_{N+1} and equation

$$\dot{x}_{N+1} = g(x, t, p).$$

Then $G = x_{N+1}(x, T, p)$. In this way, we can transform any objective function in the integral form (2) into the scalar form $g(x, T, p)$. The quadrature variables can be calculated very efficiently [15] by a staggered method in DASPK3.0; they do not enter into the Jacobian matrix.

From [6] we know that for DAE systems of index up to two (Hessenberg), asymptotic numerical stability in solving the forward problem is preserved by the backward Euler method, but only (for fully implicit DAE systems) if the discretization of the time derivative is performed 'conservatively',

which corresponds to solving an *augmented adjoint* DAE system

$$\dot{\bar{\lambda}} - F_x^* \lambda = 0,$$
$$\bar{\lambda} - F_{\dot{x}}^* \lambda = 0. \tag{7}$$

It was shown in [6] that the system (7) with respect to $\bar{\lambda}$ preserves the stability of the original system. Note that the augmented system (7) is of (one) higher index than the original adjoint system (5). This is not a problem in the implementation since the newly high-index variables do not enter into the error estimate (see Section 5.4) and it can be shown that basic DAE structures such as combinations of semi-explicit index-1 and Hessenberg index-2 are preserved under the augmentation. Also, the linear algebra is accomplished in such a way (see Section 5.1) that the matrix needed is the transpose of that required for the original systems. Thus, there are no additional conditioning problems for the linear algebra due to the use of the augmented adjoint system.

3. Evaluation of the adjoint DAE

As we have seen, the adjoint DAE must be solved backward for its solution at $t = 0$. Since not every DAE solver can take backward steps during the integration, we apply a time reversing transformation $\tau = T - t$ to the adjoint system (5). This yields

$$\dot{\lambda}_T^* F_{\dot{x}}(T - \tau, x) + \lambda_T^* \left(F_x(T - \tau, x) - \frac{\mathrm{d}F_{\dot{x}}}{\mathrm{d}t}(T - \tau, x) \right) = 0, \tag{8}$$

where $\dot{\lambda}_T^* = \mathrm{d}\lambda_T^*/\mathrm{d}\tau$, $\dot{x} = \mathrm{d}x/\mathrm{d}t$, and $\mathrm{d}F_{\dot{x}}/\mathrm{d}t$ is the total derivative of $F_{\dot{x}}$ with respect to t. If we assume that the DAE system is linear with respect to $\dot{x}$, then for any constant vector v, this term may be evaluated by

$$v \frac{\mathrm{d}F_{\dot{x}}}{\mathrm{d}t} = vF_{\dot{x}t} + (vF_{\dot{x}})_x \dot{x}. \tag{9}$$

For many applications, $F_{\dot{x}}$ is constant and Eq. (8) becomes

$$\dot{\lambda}_T^* F_{\dot{x}} + \lambda_T^* F_x(T - \tau, x) = 0. \tag{10}$$

In these cases, we do not need to evaluate the term $\mathrm{d}F_{\dot{x}}/\mathrm{d}t$. If $F_{\dot{x}}$ is time-varying, the augmented adjoint system (7), which becomes

$$\dot{\bar{\lambda}} + F_x^* \lambda = 0,$$
$$\bar{\lambda} - F_{\dot{x}}^* \lambda = 0 \tag{11}$$

under the time-reversing transformation, is used to preserve the stability.

Eqs. (10) and (11) involve matrix–vector products from the left side (referred to as vector–matrix products in the following). Although a matrix–vector product $F_x v$ can be approximated via a directional derivative finite difference method, it is difficult to evaluate the vector–matrix product vF_x directly via a finite difference method. The vector–matrix product vF_x can be written as a gradient of the function $vF(x)$ with respect to x. However, N evaluations of $vF(x)$ are required to calculate the

gradient by a finite-difference method if we do not assume any sparsity in the Jacobian. Therefore, automatic differentiation (AD) is necessary to improve the computational efficiency. A forward mode AD tool cannot compute the vector–matrix products without evaluation of the full Jacobian. It has been shown [10] that an AD tool with reverse mode can evaluate the vector-Jacobian product as efficiently as a forward mode AD tool can evaluate the Jacobian-vector product. In our implementation with DASPK3.0, we use the AD tool TAMC [10] to calculate the vector–matrix products.

ADIFOR [2] is another AD tool. Until ADIFOR3.0 is released, ADIFOR includes only the forward mode. If the SparseLinC option is used, ADIFOR can evaluate the non-zero elements of the Jacobian efficiently. After the Jacobian is computed, the vector-Jacobian product can be easily obtained. This method still requires the evaluation of a full Jacobian, which is far more expensive than a vector–matrix product.

4. Initialization of the adjoint DAE

In this section, we consider the initialization of the adjoint DAE for the scalar objective function $g(x, T, p)$. All of the adjoint DAE systems in this section are of the form (8), which means that the time reversing transformation has already been applied.

4.1. ODE systems

From Section 2, we know that the adjoint variables must satisfy initial conditions at time $t = T$. For index-0 or index-1 DAEs, $\lambda^*(T)$ satisfies $\lambda^* F_{\dot{x}}|_{t=T} = 0$. If the original DAE is of ODE standard form or regular implicit form (non-singular mass matrix $F_{\dot{x}}$), we can take $\lambda(T) = 0$. Then $\lambda_T(T)$ is given by

$$\lambda_T(T) = -\frac{\mathrm{d}\lambda}{\mathrm{d}t}|_{t=T} = \mathrm{d}\lambda/\mathrm{d}\tau|_{t=T} = (F_{\dot{x}}^*)^{-1} g_x|_{t=T}.$$

An important point to note is that computation of $\lambda(t)$ is not required for the initialization of λ_T. This will be the case also for higher-index DAEs, although there it may be less obvious.

4.2. Semi-explicit index-one DAE systems

For index-1 DAE systems of the form

$$\begin{aligned} M(t, x^d)\dot{x}^d &= f(t, x^d, x^a, p), \\ 0 &= h(t, x^d, x^a, p), \end{aligned}$$

where x^d and x^a denote the differential and algebraic solution variables, respectively, $\partial h/\partial x^a$ is non-singular, and the mass matrix $M(t, x^a)$ is a non-singular square matrix, a two-step process can be used. First we initialize the adjoint DAE for the objective function of integral form $G = \int_0^T g(x, t, p)\,\mathrm{d}t$

$$\begin{aligned} M(t, x^d)^* \dot{\lambda}^d &= (f_{x^d} + \mathrm{d}M/\mathrm{d}t - \mathrm{d}(M\dot{x}^d)/\mathrm{d}x^d)^* \lambda^d + h_{x^d}^* \lambda^a + g_{x^d}, \\ 0 &= f_{x^a}^* \lambda^d + h_{x^a}^* \lambda^a + g_{x^a} \end{aligned}$$

with $\lambda^d(T)=0$ to obtain consistent initial values for $\dot{\lambda}^d$ and λ^a. $\dot{\lambda}^d$ is the derivative of λ^d with respect to τ. During the initialization, the values of the differential variables $\lambda^d(T)$ are fixed. Second, we set $\lambda_T^d(T)=\dot{\lambda}^d(T)$ and initialize the adjoint DAE

$$M(t,x^d)^*\dot{\lambda}_T^d = (f_{x^d} + \mathrm{d}M/\mathrm{d}t - \mathrm{d}(M\dot{x}^d)/\mathrm{d}x^d)^*\lambda_T^d + h_{x^d}^*\lambda_T^a,$$
$$0 = f_{x^a}^*\lambda_T^d + h_{x^a}^*\lambda_T^a$$

for the objective function $g(x,T,p)$. The differential variables $\lambda_T^d(T)$ are fixed during the second step. Each initialization can be done easily in DASPK3.0 [13].

4.3. Fully implicit index-one DAE systems

For fully implicit index-1 DAE systems

$$F(x,\dot{x},t,p)=0,$$

the initialization can be posed as a least-squares problem for

$$A^*\dot{\lambda} + B^*\lambda = g_x^*,$$
$$A^*\lambda = 0,$$

where $A=\partial F/\partial\dot{x}|_{t=T}$ and $B=\partial F/\partial x|_{t=T}$. The solution of this problem is not currently implemented in DASPKADJOINT.

4.4. Hessenberg index-two DAE systems

For Hessenberg index-2 DAE systems

$$\dot{x}^d = f(t,x^d,x^a,p),$$
$$0 = h(t,x^d,p), \tag{12}$$

the adjoint DAE system for the objective function $g(x,T,p)$ is given by

$$\dot{\lambda}_T^{d*} = -\lambda_T^{d*}f_{x^d} - \lambda_T^{a*}h_{x^d},$$
$$0 = \lambda_T^{d*}f_{x^a}.$$

The initial values for the adjoint variable λ_T^d satisfy [6]

$$\lambda_T^d(T) = P^*(g_{x^d}^* + f_{x^d}^*\lambda^d - \dot{h}_{x^d}^*(f_{x^a}^*h_{x^d}^*)^{-1}g_{x^a}^*)|_{t=T} \tag{13}$$

or

$$\lambda_T^{d*}(T) = (g_{x^d} + \lambda^{d*}f_{x^d} - g_{x^a}(h_{x^d}f_{x^a})^{-1}\dot{h}_{x^d})P|_{t=T}, \tag{14}$$

where $\lambda^{d*}=-g_{x^a}(h_{x^d}f_{x^a})^{-1}h_{x^d}$ is the adjoint variable for the objective function $G=\int_0^T g(t,x,p)\,\mathrm{d}t$, $P=I-f_y(h_xf_y)^{-1}h_x$ is a projection matrix for the original index-2 system, and $\dot{h}_{x^d}$ is the total

derivative of h_{x^d} with respect to t, $\dot{h}_{x^d} = h_{x^d t} + (h_{x^d}\dot{x}^d)_{x^d}$. Note that f_{x^d}, f_{x^a}, h_{x^d} and $\dot{h}_{x^d}$ are matrices, and the initialization in this case is much more complicated. In the following, with the help of an AD tool and the options of DASPK3.0, we give a matrix-free implementation.

The matrix-free initialization procedure can be split into four steps. First we compute $\lambda^{d*}(T) = -g_{x^a}(h_{x^d} f_{x^a})^{-1} h_{x^d}|_{t=T}$ by solving the following initialization problem:

$$\begin{aligned} \dot{\lambda}_1^{d*} &= \lambda_1^{d*} f_{x^d} + \lambda_1^{a*} h_{x^d}, \\ 0 &= \dot{\lambda}_1^{d*} f_{x^a} - \lambda_1^{d*} \dot{f}_{x^a} - g_{x^a} \end{aligned} \tag{15}$$

for $\dot{\lambda}_1^d(T)$ and $\lambda_1^q(T)$ with $\lambda_1^d(T) = 0$ fixed. In (15), $\dot{f}_{x^a}$ represents the total derivative of f_{x^a} with respect to t. Note that $\lambda_1^{a*}(T) = g_{x^a}(h_{x^d} f_{x^a})^{-1}|_{T=T}$, $\lambda^{d*}(T) = -\dot{\lambda}_1^{d*}(T) = -g_{x^a}(h_{x^d} f_{x^a})^{-1} h_{x^d}|_{t=T}$. If $g_{x^a} = 0$, then $\lambda^{d*}(T) = 0$.

In step 2, we calculate

$$v_1 = g_{x^d} + \lambda^{d*} f_{x^d} - g_{x^a}(h_{x^d} f_{x^a})^{-1} \dot{h}_{x^d}|_{t=T} = g_{x^d} + \lambda^{d*} f_{x^d} - \lambda_1^{a*} \dot{h}_{x^d}|_{t=T}. \tag{16}$$

If $g_{x^a} = 0$, then $v_1 = g_{x^d}|_{t=T}$ and we can go directly to step 3. $\lambda^{d*} f_{x^d}$ can be calculated easily by an AD tool with reverse mode. The calculation of $\lambda_1^{a*}\dot{h}_{x^d}$ is more troublesome. If h_{x^d} is a constant matrix then $\lambda_1^{a*}\dot{h}_{x^d} = 0$. Otherwise, an AD tool with a combination of reverse and forward modes is used to evaluate $\lambda_1^{a*}\dot{h}_{x^d}$. $\lambda_1^{a*} h_{x^d}$ is first calculated by a reverse mode AD tool. Then the vector–matrix product is differentiated explicitly by a forward mode AD-tool with respect to t and x^d.

In step 3, we calculate the initial values of λ_T^d

$$\lambda_T^{d*}(T) = v_1 P|_{t=T} = v_1(I - f_{x^a}(h_{x^d} f_{x^a})^{-1} h_{x^d})|_{t=T} \tag{17}$$

by initializing the following system:

$$\begin{aligned} \dot{\lambda}_2^{d*} &= \lambda_2^{d*} f_{x^d} + \lambda_2^{a*} h_{x^d} + v_1, \\ 0 &= \dot{\lambda}_2^{d*} f_{x^a} - \lambda_2^{d*} \dot{f}_{x^a} \end{aligned} \tag{18}$$

with $\lambda_2^{d*}(T) = 0$ fixed. Noting that $\lambda_2^{a*}(T) = -v_1 f_{x^a}(h_{x^d} f_{x^a})^{-1}|_{t=T}$ and $\dot{\lambda}_2^{d*} = v_1(I - f_{x^a}(h_{x^d} f_{x^a})^{-1} h_{x^d})|_{t=T}$, we set $\lambda_T^d(T) = \dot{\lambda}_2^{d*}(T)$.

In step 4, we initialize the adjoint system for the objective function $g(x, T, p)$ by fixing the value of $\lambda_T^{d*}(T)$. The index-2 constraints must be differentiated during the initialization, which results in

$$\begin{aligned} \dot{\lambda}_T^{d*} &= \lambda_T^{d*} f_{x^d} + \lambda_T^{a*} h_{x^d}, \\ 0 &= \dot{\lambda}_T^{d*} f_{x^a} - \lambda_T^{d*} \dot{f}_{x^a}, \end{aligned} \tag{19}$$

where $\lambda_T^d(T)$ is fixed and $\dot{\lambda}_T^d(T)$ and $\lambda_T^a(T)$ are computed.

The DAE systems (15) and (18) have the same format except for the forcing terms. Therefore, we can solve them efficiently with the same algorithm. The second equation of the adjoint DAE

system (19) is actually the derivative of the index-2 constraint $\lambda^d f_{x^a}$ with respect to reversed time τ. DASPKADJOINT has a mechanism to differentiate the index-2 constraints and then to perform the initialization for index-2 DAE systems in DASPK3.0 [15].

For index-2 DAE systems which are not explicitly in Hessenberg form, but which have a non-singular square mass matrix, i.e.,

$$\begin{aligned} M(t,x^d)\dot{x}^d &= f(t,x^d,x^a,p),\\ 0 &= h(t,x^d,p), \end{aligned} \tag{20}$$

where $M(t,x^d)$ is a non-singular square matrix, the adjoint system is

$$\begin{aligned} M(t,x^d)^*\dot{\lambda}_T^d &= (f_{x^d}+\dot{M}-(M\dot{x}^d)_{x^d})^*\lambda_T^d + h_{x^d}^*\lambda_T^a,\\ 0 &= f_{x^a}^*\lambda_T^d. \end{aligned} \tag{21}$$

The initial values for the adjoint variables λ_T^d satisfy

$$\begin{aligned} \lambda_T^{d*} &= \left(\dot{\lambda}^d - g_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}\frac{\mathrm{d}(h_{x^d}M^{-1})}{\mathrm{d}t}\right)P\\ &= ((g_{x^d}+\lambda^{d*}(f_{x^d}+\dot{M}-(M\dot{x}^d)_{x^d}))M^{-1} - g_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}(\dot{h}_{x^d}M^{-1}-h_{x^d}M^{-1}\dot{M}M^{-1}))P\\ &= (g_{x^d}+\lambda^{d*}(f_{x^d}-(M\dot{x}^d)_{x^d}) - g_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}\dot{h}_{x^d})M^{-1}P\\ &= (g_{x^d}-\lambda^{d*}F_{x^d}-\lambda_1^{a*}\dot{h}_{x^d})M^{-1}P\\ &= v_1M^{-1}P \end{aligned}$$

at $t=T$, where $\lambda^{d*}=-g_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}h_{x^d}M^{-1}$, $P=I-f_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}h_{x^d}M^{-1}$, $F=M\dot{x}^d-f(x^d,x^a,p)$, $\lambda_1^{a*}=g_{x^a}(h_{x^d}M^{-1}f_{x^a})^{-1}$, and $v_1=g_{x^d}-\lambda^{d*}F_{x^d}-\lambda_1^{a*}\dot{h}_{x^d}$. The above initialization procedures for Hessenberg form can still be used by replacing the adjoint DAE system with (21).

5. Implementation

In this section we outline some of the issues for implementation of the adjoint method for DAE sensitivity analysis. Although any DAE solver could in principle be used for the forward and backward integration of the adjoint DAE system, we describe the implementation via our software DASPK3.0 [15].

In the adjoint system (5) and the sensitivity calculation (6), the derivatives F_x, $F_{\dot{x}}$ and F_p may depend on the state variables x, which are the solutions of the original DAEs. Ideally, the adjoint DAE (5) should be coupled with the original DAE and solved together as we did in the forward sensitivity method. However, in general it is not feasible to solve them together because the original DAE may be unstable when solved backward. Alternatively, it would be extremely inefficient to solve the original DAE forward any time we need the values of the state variables.

The implementation of the adjoint sensitivity method consists of three major steps. First, we must solve the original ODE/DAE forward to a specific output time T. Second, at time T, we compute the consistent initial conditions for the adjoint system. The consistent initial conditions must satisfy the boundary conditions of (3). Finally, we solve the adjoint system backward to the start point, and calculate the sensitivities.

With enough memory, we can store all of the necessary information about the state variables at each time step during the forward integration and then use it to obtain the values of the state variables by interpolation during the backward integration of the adjoint DAEs. For example, we can store x and $\dot{x}$ at each time step during the forward integration and reconstruct the solution at any time by cubic Hermite interpolation [1] during the backward integration. The memory requirements for this approach are proportional to the number of time steps and the dimension of the state variables x, and are unpredictable because the number of time steps varies with different options and error tolerances of the ODE/DAE solver.

To reduce the memory requirements and also make them predictable, we use a two-level checkpointing technique. First we set up a checkpoint after every fixed number of time steps during the forward integration of the original DAE. Then we recompute the forward information between two consecutive checkpoints during the backward integration by starting the forward integration from the checkpoint. This approach needs to store only the forward information at the checkpoints and at fixed number of times between two checkpoints.

In the implementation we allocated a special buffer to communicate between the forward and backward integration. The buffer is used for two purposes: to store the necessary information to restart the forward integration at the checkpoints, and to store the state variables and derivatives at each time step between two checkpoints for reconstruction of the state variable solutions during the backward integration.

In order to obtain the fixed number of time steps between two consecutive checkpoints, the second forward integration should make exactly the same adaptive decisions as the first pass if it restarts from the checkpoint. Therefore, the information saved at each checkpoint should be enough that the integration can repeat itself. In the case of DASPK3.0, the necessary information includes the order and stepsize for the next time step, the coefficients of the BDF formula, the history information array of the previous k time steps, the Jacobian information at the current time, etc. To avoid storing Jacobian data (which is much larger than other information) in the buffer, we enforce a reevaluation of the iteration matrix at each checkpoint during the first forward integration.

If the size of the buffer is specified, the maximum number of time steps allowed between two consecutive checkpoints and the maximum number of checkpoints allowed in the buffer can be easily determined. However, the total number of checkpoints is problem-dependent and unpredictable. It is possible that the number of checkpoints is also too large for some applications to be held in the buffer. We then write the data of the checkpoints from the buffer to a disk file and reuse the buffer again. Whenever we need the information on the disk file, we can access it from the disk. We assume that the disk is always large enough to hold the required information.

[1] We could of course consider basing the interpolant on the interpolating polynomial underlying the BDF formula, but this is more complicated, requires more storage, and it not as smooth.

5.1. Jacobian or preconditioner evaluation and Newton solver

For an implicit ODE/DAE solver (e.g., BDF, SIRK), a Jacobian (or preconditioner) is required for solving the non-linear system at each time step. The Jacobian for the adjoint DAE is exactly the transpose of the Jacobian for the original DAE when the mass matrix $F_{\dot{x}}$ is constant. Therefore, we can evaluate the Jacobian of the original DAE and transpose it to obtain the Jacobian for the adjoint DAE.

If $F_{\dot{x}}$ depends on either t or x, we can use the adjoint DAE (7) or (11) for discretization. The Jacobian for system (11) is

$$\begin{pmatrix} \alpha I & F_x^* \\ I & -F_{\dot{x}}^* \end{pmatrix},$$

where α is a scalar coefficient in DASPK3.0. If we solve only for λ, then the second column is multiplied by $-\alpha$ and added to the first column, which yields the Jacobian for λ only, $\alpha F_{\dot{x}}^* + F_x^*$, which remains the transpose of the original Jacobian for the forward integration.

The adjoint system is a linear time-varying system. The Newton solver can converge in one iteration if the Jacobian is up to date. In DASPK3.0, the convergence test of the Newton solver is

$$\frac{\rho}{1-\rho}\|y^{(m+1)} - y^{(m)}\| < 0.33, \tag{22}$$

where the rate of convergence ρ is given by

$$\rho = \left(\frac{\|y^{(m+1)} - y^{(m)}\|}{\|y^{(1)} - y^{(0)}\|}\right)^{1/m}.$$

If the Newton solver converges in one iteration, ρ can be very small and is passed on to future time steps. The near-zero ρ can make (22) satisfied even if $\|y^{(m+1)} - y^{(m)}\|$ is large. The large value of $\|y^{(m+1)} - y^{(m)}\|$ can yield bad results if the Jacobian is not current. This is a deficiency in the convergence test of DASPK3.0 (and previous versions), but to our knowledge it has never before been the source of difficulty because unlike the adjoint DAE, the vast majority of DAEs solved in practice are non-linear.

There are two options to fix the convergence test for the adjoint system: force a Jacobian evaluation on every step, or force a recalculation of the convergence rate on every step. Because a Jacobian evaluation takes much more time than a function evaluation, we force a recalculation of the convergence rate on every step for the adjoint method using DASPK3.0. The initial value of ρ is always 0.99.

5.2. Krylov iterative method

For the Krylov iterative method, we need to evaluate the matrix–vector product

$$u = (\alpha F_{\dot{x}}^* + F_x^*)v. \tag{23}$$

Since we already have the mechanism to evaluate $vF_{\dot{x}}$ and vF_x when we evaluate the adjoint DAE, (23) can be evaluated directly.

A preconditioner must be provided by the user for the Krylov iterative method in DASPK3.0. The preconditioner provided by the user usually works well for the forward integration of the state variables and for the forward sensitivity computation. However, it is not always a simple matter to apply this preconditioner for the adjoint system. Because the Jacobian for the adjoint system is the transpose of the forward Jacobian, a transpose of the preconditioner is needed. If a matrix-free method has been used to construct the preconditioner for the forward system, it is difficult to transpose the preconditioner and apply it to the adjoint system. On the other hand, if the Jacobian matrix is constructed during computation of the preconditioner as, for example, with the incomplete LU factorization (ILU) preconditioner, the preconditioner for the adjoint system can be easily obtained by transposing the Jacobian first and then doing the incomplete LU factorization.

5.3. Sensitivity calculation

The integral in Eq. (6) can be computed either outside DASPK after each time step or as a quadrature variable during the solution of the adjoint DAE. If the quadrature variable method is chosen, we append a variable, say λ_{N+1}, and an equation

$$\dot{\lambda}_{N+1} = \lambda_T^* F_p$$

to the original adjoint system. Then

$$\lambda_{N+1}|_{t=0} = \int_0^T \lambda_T^* F_p \, \mathrm{d}t.$$

λ_{N+1} is called a quadrature variable in DASPK3.0 and can be calculated efficiently [15] without participating in Newton iterations and Jacobian evaluations via a staggered method. The other terms in (6) can be easily obtained: $\lambda^* F_p$ and $\lambda_T F_{\dot{x}}$ can be computed by an AD tool with reverse mode.

$(x_0)_p$ is the Jacobian matrix of the initial condition x_0 with respect to the sensitivity parameter p. When p is the initial condition for an ODE (or implicit ODE), $(x_0)_p = I$ is the identity matrix. For an index-1 or index-2 system, $(x_0)_p$ must be consistent with the algebraic constraints and/or any hidden constraints. $(x_0)_p$ can be input either by the user or evaluated by an AD tool. When the number of parameters is large, the matrix $(x_0)_p$ is huge. However, the number of non-zero elements in $(x_0)_p$ may be small. A sparse format to store and compute $(x_0)_p$ is necessary.

For the Hessenberg index-2 DAE system (12), the value of λ^{a*} is required when $h_p \neq 0$, because $\lambda^{a*} h_p$ should be calculated in the sensitivity (6). To solve for λ^{a*} correctly in (16), we need the value of $\mathrm{d}g_{x^a}/\mathrm{d}t$, which is difficult for the user to provide. However, there is an additional term for the sensitivity (6) if $h_p \neq 0$, which is $\mathrm{d}(-g_{x^a}(h_{x^d} f_{x^a})^{-1} h_p)/\mathrm{d}t$. Then the combination of the two terms will be

$$\begin{aligned}
&(\lambda^{d*} f_p + \lambda^{a*} h_p) + \frac{\mathrm{d}}{\mathrm{d}t}(-g_{x^a}(h_{x^d} f_{x^a})^{-1} h_p) \\
&\quad = \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + \left(\lambda^{a*} - \frac{\mathrm{d}g_{x^a}}{\mathrm{d}t}(h_{x^d} f_{x^a})^{-1} + g_{x^a}(h_{x^d} f_{x^a})^{-1} \left(\dot{h}_{x^d} f_{x^a} + h_{x^d} \dot{f}_{x^a} \right) (h_{x^d} f_{x^a})^{-1} \right) h_p \\
&\quad = \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + (\lambda^{a*} h_{x^d} - \dot{\lambda}^{d*} + \lambda_1^{a*} \dot{h}_{x^d}) f_{x^a} (h_{x^d} f_{x^a})^{-1} h_p \\
&\quad = \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + \lambda_2^{a*} h_p.
\end{aligned}$$

where $\lambda^{d*} f_{x^a} + g_{x^a} = 0$ is used and $\lambda_1^{a*} = -g_{x^a}(h_{x^d} f_{x^a})^{-1}$ and $\lambda_2^{a*} = -(g_{x^d} + \lambda^{d*} f_{x^d} - \lambda_1^{a*} \dot{h}_{x^d}) f_{x^a}(h_{x^d} f_{x^a})^{-1}$ can be obtained from Eqs. (15) and (18), respectively. It can be verified that the last equation is also valid for the case of non-Hessenberg form given by (20).

5.4. Error estimation

Two of the most important decisions that an adaptive DAE solver must make on each step are whether to accept the results of the current step and what stepsize should be used on the next step. Both of these decisions are based on the error estimate. DASSL/DASPK estimates the local truncation error, but it is not obvious how this should be implemented for the adjoint solution.

To get a better understanding, consider applying the implicit Euler method to the augmented adjoint system (11),

$$\begin{aligned} &\frac{\bar{\lambda}_{n+1} - \bar{\lambda}_n}{h_{n+1}} + F_x^* \lambda_{n+1} = 0, \\ &\bar{\lambda}_{n+1} - F_{\dot{x}}^* \lambda_{n+1} = 0. \end{aligned} \tag{24}$$

The true solution to (11) satisfies

$$\begin{aligned} &\frac{\bar{\lambda}(t_{n+1}) - \bar{\lambda}(t_n) - (h_{n+1}^2/2)\bar{\lambda}''(\xi)}{h_{n+1}} + F_x^* \lambda(t_{n+1}) = 0, \\ &\bar{\lambda}(t_{n+1}) - F_{\dot{x}}^* \lambda(t_{n+1}) = 0. \end{aligned} \tag{25}$$

Subtracting (25) from (24), we obtain

$$\frac{\bar{e}_{n+1} - \bar{e}_n + (h_{n+1}^2/2)\bar{\lambda}''(\xi)}{h_{n+1}} + F_x^* e_{n+1} = 0,$$

where $\bar{e}_n = \bar{\lambda}_n - \bar{\lambda}(t_n)$ and $e_n = \lambda_n - \lambda(t_n)$. Thus, the local truncation error (the amount by which the true solution fails to satisfy the difference formula) depends on $\bar{\lambda}''$ rather than on λ''. Therefore, the error estimate should be based on $\bar{\lambda}$. Since the errors in the algebraic variables on previous time steps do not directly influence the errors in any of the variables at the current time [17], we can consider deleting the λ from the error estimate in order to promote the smooth operation of a code. However, the value of λ is important and determines the accuracy of the sensitivities. So for the λ variables which are index-1 in the augmented adjoint system, we opted to include them and to exclude those which are index-2.

5.5. Efficiency of the adjoint sensitivity method

In this subsection we compare the adjoint sensitivity method with the forward sensitivity method. Suppose the cost of the forward integration for N state variables is C_f, the cost of the forward integration for N sensitivity variables is C_{fs}, the cost of the backward integration for N adjoint variables is C_{bs}, the number of parameters is n_p, and the number of objective functions is n_f.

Then our implementation of the adjoint sensitivity method includes two forward integrations and one backward integration.[2] The total cost is roughly

$$C_{asm} = 2C_f + C_{bs}^* + (n_f - 1)C_{bs},$$

where C_{bs}^* denotes the cost for the first objective function, which is more costly than the others because it involves the Jacobian evaluation. The total cost for the forward sensitivity method to perform the equivalent computation is roughly

$$C_{fsm} = C_f + n_p C_{fs}.$$

The computational efficiency and memory requirements for the forward sensitivity method are roughly proportional to the number of sensitivity parameters and are insensitive to the number of objective functions. For the adjoint sensitivity method, the computational efficiency and memory requirements are proportional to the number of objective functions and are insensitive to the number of sensitivity parameters. Thus the two methods are complementary. The adjoint sensitivity method is advantageous over the forward sensitivity method when the number of sensitivity parameters is large and the number of objective functions is small.

The adjoint sensitivity method has a disadvantage that it can only compute the sensitivity at a specific output time. Unlike the forward sensitivity method, the intermediate results of the adjoint variables have no physical meaning.

6. Numerical experiments

In this section we give some examples to demonstrate the effectiveness and efficiency of the adjoint sensitivity method as implemented in DASPKADJOINT. In all of our examples, the tolerance for the adjoint variables has been taken to be double the tolerance for the state variables, because we have found that it is helpful in the solution of the adjoint equations to have solved for the state variables with slightly greater accuracy than we request for the adjoint variables. The integration methods used are the direct method (D) and Krylov method (K). The sensitivity methods are the forward sensitivity method (F), using the staggered corrector option, from DASPK3.0 [14] and the adjoint sensitivity method (A). Therefore, we use AD to represent the adjoint direct method, AK to represent the adjoint Krylov method, FD to represent the forward direct method, and FK to represent the forward Krylov method. The computation was performed on a Linux machine with Pentium III 450 MHZ CPU.

6.1. ODE case

6.1.1. 2-D heat equation

We first consider the heat equation

$$u_t = p_1 u_{xx} + p_2 u_{yy} \tag{26}$$

[2] This assumes that checkpointing is used. If checkpointing is not needed, it requires only one forward integration.

Table 1
Results for heat equation example. NP is the number of sensitivity parameters

METH	NP	g_1 w.r.t. p_1	g_2 w.r.t. p_1	RWORK size	IWORK size	CPU
FD	20	−15.21783	−2.72675	669025	10630	44.15
FD	10	−15.21783	−2.72675	457205	10630	24.40
AD	1766	−15.21831	−2.72685	563563	12462	21.11
FK	10	−15.21782	−2.72675	329420	83868	15.39
AK	1766	−15.21794	−2.72667	235388	167751	6.63

posed on the 2-D unit square with zero Dirichlet boundary conditions. An $M+2$ by $M+2$ mesh is used 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 $u_t = 0$. The discrete values of u form a vector U, ordered first by x, then by y. The result is an ODE system $G(t, U, U') = 0$ of size $NEQ = (M+2) \times (M+2)$. Initial conditions are posed as

$$u_{t=0} = 16x(1-x)y(1-y).$$

The sensitivity parameters consist of $p_1 = p_2 = 1.0$ and the initial values $u_{t=0}$. The problem was solved previously by DASPK3.0 with $M = 40$ in [15]. To compute the sensitivities by the adjoint method, we used the time interval $[0, 0.16]$, and the error tolerances for DASPK3.0 were taken as $\mathrm{RTOL} = \mathrm{ATOL} = 1.0\mathrm{e}-5$. The size of the buffer was set to such a number that it allows nine time steps between two consecutive checkpoints, and the maximum number of checkpoints the buffer allows was set to 3. There are a total of 10 checkpoints during the forward integration of the state variables and the information at the checkpoints has to be written to the disk file three times. For the direct method, we used the ADIFOR option with SparseLinC 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 also evaluated by ADIFOR with the SparseLinC option. We compared the results with that of the forward sensitivity method where the sensitivity residuals are evaluated by ADIFOR with the seed matrix option. Due to the memory restriction, we used only 20 sensitivity parameters (including p_1 and p_2) in the forward sensitivity method. For comparison, we used two objective functions:

$$g_1 = \sum_{1}^{\mathrm{NEQ}} u_i^2,$$

$$g_2 = \int_0^T \sum_{1}^{\mathrm{NEQ}} u_i \, \mathrm{d}t,$$

where g_2 is treated as a quadrature variable. Table 1 gives the results of the adjoint and forward methods.

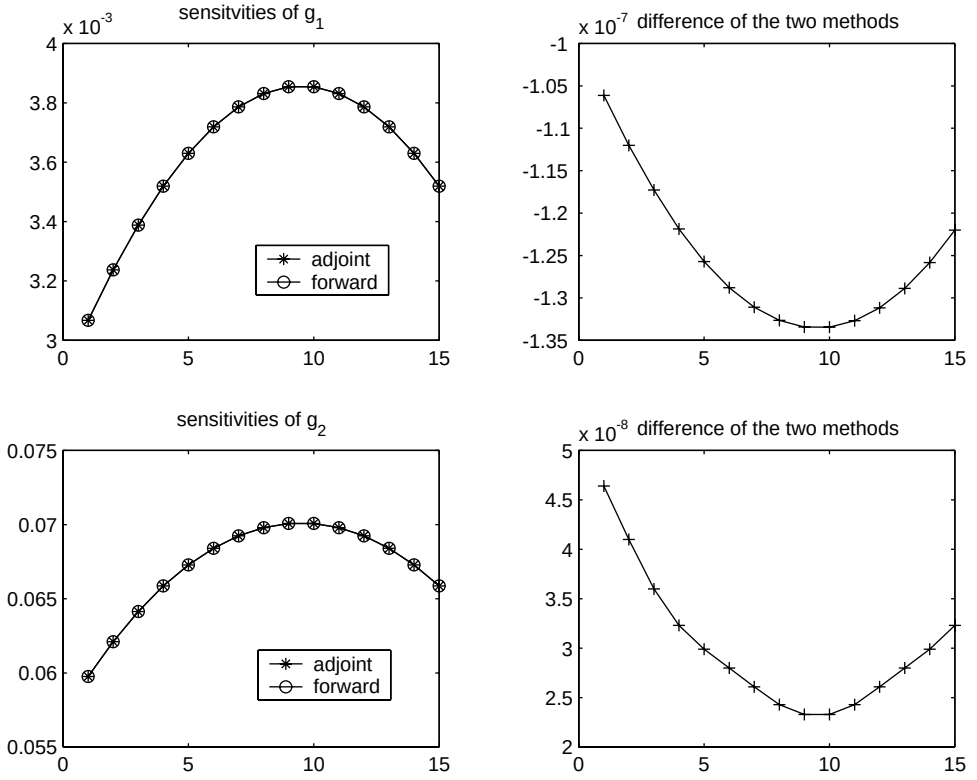

Fig. 1. Sensitivities of the objective functions with respect to the initial conditions for the heat equation example. The x-axis represents the order number of the points.

Fig. 1 shows the sensitivities of the objective functions with respect to the initial conditions. We chose the points between 13 and 27 in the 20th row on the 42×42 mesh as our sample points for the figure.

6.1.2. *Index-0 example with state-dependent mass matrix*

We also tested the case when the coefficient matrix $F_{\dot{x}}$ is not constant. Consider the system

$$\begin{pmatrix} y_1 & y_2 \\ -y_2 & y_1 \end{pmatrix} \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -(y_1^2 + y_2^2) \end{pmatrix} \tag{27}$$

with initial conditions $y_1(0) = 0,\ y_2(0) = 1$. We used the time interval [0,1.57] and the objective function $g(y) = y_1 + y_2$ in the test. The sensitivity parameters were taken to be $y_1(0)$ and $y_2(0)$. The augmented adjoint system for g is

$$\dot{\lambda} - \begin{pmatrix} y_1 & -y_2 \\ y_2 & y_1 \end{pmatrix} \lambda = 0,$$

Table 2
Results for example (27). $\mathrm{Err}_{\max}$ is the maximum error in λ. NSTP is the number of time steps, NRES is the number of function evaluations, NJAC is the number of Jacobian evaluations, NETF is the number of error test failures and NNL is the number of non-linear iterations. There are no convergence test failures in any of the tests

Error control	Recalculate ρ	NSTP	NRES	NJAC	NETF	NNL	$\mathrm{Err}_{\max}$
$\bar{\lambda}$	No	64	245	31	7	121	0.6025
λ, $\bar{\lambda}$	No	144	383	31	7	259	8.0e-7
λ	No	2662	28486	4174	2660	11790	5.65e-4
$\bar{\lambda}$	Yes	61	254	28	7	142	7.0e-7
λ, $\bar{\lambda}$	Yes	86	317	28	7	205	1.4e-7
λ	Yes	2662	28487	4174	2660	11791	5.65e-4

$$\dot{\bar{\lambda}} + \begin{pmatrix} y_1' & y_2' + 2y_1 \\ y_2' & -y_1' + 2y_2 \end{pmatrix} \lambda = 0.$$

We used the error tolerances $\mathrm{ATOL} = 10^{-9}$, $\mathrm{RTOL} = 10^{-7}$ for DASPK3.0 in the test. The results were

METH	g w.r.t. $y_1(0)$	g w.r.t. $y_2(0)$
FD	−0.999204383	1.00079725
AD	−0.999203795	1.00079653
True solution	−0.999203356	1.00079601

This example illustrates the need for the modifications to the DASPK3.0 error test and Newton convergence test strategies described earlier for solving the augmented adjoint system. It can be shown that when $y_1 = \sin(t)$ and $y_2 = \cos(t)$, $\dot{\lambda} = 0$. If we base the error estimate only on λ, which may at first glance seem to be natural, it results in a large number of error test failures. The reason is that according to the dynamics of λ, DASPK3.0 tries with this error estimate to double the stepsize at almost every time step. However, the local truncation error is determined by $\bar{\lambda}$ instead of λ, so these large stepsizes fail based on the dynamics of $\bar{\lambda}$. We also observed that if we did not force a recalculation of the convergence rate ρ during the Newton iteration, a large error can occur in λ if we base the error test only on $\bar{\lambda}$. The results are good and the code operates efficiently when we base the error test on both λ and $\bar{\lambda}$, or base the error test only on $\bar{\lambda}$ and recalculate the convergence rate. Table 2 gives the results of the different options for the error control and convergence tests.

6.2. Index-1 DAE case

6.2.1. 2-D food web problem

We consider 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

$$c_t^1 = f_1(x, y, t, c) + d_1(c_{xx}^1 + c_{yy}^1),$$

$$0 = f_2(x, y, t, c) + d_2(c_{xx}^2 + c_{yy}^2)$$

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, d_i are:

$$a_{ii} = -1, \quad i = 1, 2;$$

$$a_{12} = -0.5 \times 10^{-6}, \quad a_{21} = 10^4;$$

$$a_{ij} = e, \quad i > p \text{ and } j < p;$$

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

$$d_1 = 1, \quad d_2 = 0.05.$$

The domain is the unit square $0 \leqslant x, y \leqslant 1$. The boundary conditions are of Neumann type with normal derivative equal to 0. The PDEs are discretized by central differencing on an M by M mesh. We have taken $M = 20$. Therefore, the resulting DAE system has size $\mathrm{NEQ} = 2M^2 = 800$. The DASPK3.0 tolerances used were $\mathrm{RTOL} = \mathrm{ATOL} = 10^{-5}$.

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

$$c_0^1 = 10 + (16x(1 - x)y(1 - y))^2,$$

$$c_0^2 = 100,$$

which do not satisfy the constraint equations. Therefore, a consistent initialization is required. For comparison, we also took the initial values c_0^1 as sensitivity parameters. Unlike the ODE case, we cannot take both c_0^1 and c_0^2 as independent sensitivity parameters, because c_0^2 are index-1 variables and they depend on c_0^1. We used the time interval $[0, 5]$, and tolerances for DASPK3.0 of $\mathrm{RTOL} = \mathrm{ATOL} = 10^{-5}$. For the adjoint method, the size of the buffer was set to such a number that it allows 12 time steps between two consecutive checkpoints. The maximum number of checkpoints the buffer allows was set to 4. There are a total of 10 checkpoints during the forward integration of the state variables, and the information at the checkpoints has to be written to the disk file twice. We used two identical objective functions

$$g_1 = \sum_{1}^{\mathrm{NEQ}} u_i^2,$$

$$g_2 = g_2.$$

Table 3 gives the results of the adjoint and forward methods. Note that the storage requirement (RWORK) and the CPU time for the forward sensitivity method are proportional to the number of sensitivity parameters, whereas for the adjoint method they remain the same.

Table 3
Results for food-web problem. NP is the number of sensitivity parameters

METH	NP	g_1 w.r.t. p_1	g_1 w.r.t. p_2	RWORK size	IWORK size	CPU
FD	20	6467.01	3287.73	312890	4840	26.77
FD	10	6467.01	3287.73	208870	4840	14.73
FD	2	6467.01	3287.73	125654	4840	6.09
AD	402	6467.12	3287.79	249350	7313	13.14

6.2.2. Index-1 example with state-dependent mass matrix

We also tested a simple index-1 example with non-constant matrix $F_{\dot{x}}$. The equations are

$$\begin{pmatrix} y_2 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} -y_2(y_2-1) \\ y_2 - y_1 - 1 \end{pmatrix} \tag{28}$$

with initial values $y_1(0)=1$ and $y_2(0)=2$. We used the time interval [0,1] and the objective function $g(y) = y_1 + y_2$. We cannot select both $y_1(0)$ and $y_2(0)$ as sensitivity parameters for a well-posed problem. Thus we chose $y_1(0)$ as the sensitivity parameter. The DASPK3.0 error tolerances are $\mathrm{ATOL} = 10^{-9}$, $\mathrm{RTOL} = 10^{-7}$. The results are

METH	g w.r.t. $y_1(0)$
FD	0.73575887
AD	0.73575898
True solution	0.73575882

6.3. Index-2 DAE case

We consider an index-2 DAE from mechanics. This problem is selected from the set of initial value test problems [12]. It is of the form

$$M\frac{\mathrm{d}y}{\mathrm{d}t} = f(y), \quad y(0) = y_0, \quad y'(0) = y_0'$$

with $y, f \in R^{160}$, $t \in [0, 1000]$. M is a constant mass matrix given by

$$\begin{pmatrix} I_{120} & 0 \\ 0 & 0 \end{pmatrix},$$

where I_{120} is the identity matrix of dimension 120. For the definition of the function f, we refer to [12]. The first 120 components are of index-0, the last 40 of index 2.

The components $y_{0,i}$ of the initial vector y_0 are defined by

$$\begin{pmatrix} y_{0,3(j-1)+1} \\ y_{0,3(j-1)+2} \\ y_{0,3(j-1)+3} \end{pmatrix} = \begin{pmatrix} \cos(\omega_j)\cos(\beta_j) \\ \sin(\omega_j)\cos(\beta_j) \\ \sin(\beta_j) \end{pmatrix} \quad \text{for } j = 1,\ldots,20,$$

where

$$\beta_j = \alpha_1\pi \quad \text{and} \quad \omega_j = \frac{2j}{3}\pi + \alpha_2\pi \quad \text{for } j = 1,\ldots,3,$$

$$\beta_j = \alpha_3\pi \quad \text{and} \quad \omega_j = \frac{2(j-3)}{7}\pi + \alpha_4\pi \quad \text{for } j = 4,\ldots,10,$$

$$\beta_j = \alpha_5\pi \quad \text{and} \quad \omega_j = \frac{2(j-10)}{6}\pi + \alpha_6\pi \quad \text{for } j = 11,\ldots,16,$$

$$\beta_j = \alpha_7\pi \quad \text{and} \quad \omega_j = \frac{2(j-17)}{4}\pi + \alpha_8\pi \quad \text{for } j = 11,\ldots,16.$$

For the remainder of the initial conditions, the reader is referred to [12]. The vector α_i contains the sensitivity parameters, with nominal values

$$\alpha_1 = \tfrac{3}{8}, \quad \alpha_2 = \tfrac{1}{13}, \quad \alpha_3 = \tfrac{1}{8}, \quad \alpha_4 = \tfrac{1}{29},$$

$$\alpha_5 = -\tfrac{2}{15}, \quad \alpha_6 = \tfrac{1}{7}, \quad \alpha_7 = -\tfrac{3}{10}, \quad \alpha_4 = \tfrac{1}{17}.$$

The origin of the problem is to compute the elliptic Fekete points. For any configuration $x := (x_1, x_2, \ldots, x_N)^{\mathrm{T}}$ in a unit sphere in $\mathfrak{R}^3$, the points $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N$ are called the elliptic Fekete points of order N if the function

$$V(x) := \prod_{i<j} \|x_i - x_j\|_2$$

reaches its global maximum at x. This optimization problem can be formulated as an index-2 DAE [12]. Since it is a global optimization problem, the value of $V(x)$ at the final time should not depend on the initial conditions, i.e., the sensitivity of $V(x)$ with respect to the sensitivity parameters α_i should be zero or close to zero. The buffer size for the adjoint method was set to be large enough to hold the information for 30 time steps and 10 checkpoints. There are a total of eight checkpoints. Thus no temporary disk file was written.

We first computed the sensitivities by the forward sensitivity method (see Table 4). Then the adjoint sensitivity method was used. The tolerances for both methods were $\mathrm{RTOL} = \mathrm{ATOL} = 10^{-4}$. Table 4 gives the results of the adjoint and forward methods. Note that the results for the adjoint method are not as good as that of the forward sensitivity method. However, they are within the error tolerances. The CPU time used for the forward method was 11.49, and for the adjoint method 6.78. The CPU time will be almost the same for the adjoint method if more sensitivity parameters are considered, whereas it will increase for the forward method.

Table 4
Results for Fekete problem

Sensitivity	FD	AD
V_{α_1}	−3.7873e-11	6.3065e-4
V_{α_2}	0	4.8950e-5
V_{α_3}	4.4342e-11	2.5240e-4
V_{α_4}	0	2.4969e-5
V_{α_5}	4.4272e-11	−9.3772e-4
V_{α_6}	−3.7543e-11	−1.4526e-4
V_{α_7}	−3.8355e-11	5.4515e-4
V_{α_8}	−3.7363e-11	1.0331e-4

Acknowledgements

The authors would like to thank Muruhan Rathinam and Radu Serban for their interesting discussions and useful comments.

References

[1] U.M. Ascher, L.R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, Philadelphia, PA, 1998.
[2] C. Bischof, A. Carle, G. Corliss, A. Griewank, P. Hovland, ADIFOR-Generating derivative codes from Fortran programs, Scientific Programming 1 (1992) 11–29.
[3] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, 2nd 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. (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 (5) (1998) 1495–1512.
[6] Y. Cao, S. Li, L. Petzold, R. Serban, Adjoint sensitivity analysis for differential-algebraic equations: the adjoint DAE system and its numerical solution, SIAM J. Sci. Comput., to appear.
[7] M. Caracotsios, W.E. Stewart, Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations, Compu. Chem. Eng. 9 (4) (1985) 359–365.
[8] R.M. Errico, What is an adjoint model? Bulletin of the American Meteorological Society 78 (1997) 2577–2591.
[9] 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.
[10] R. Giering, T. Kaminski, Recipes for adjoint code construction, ACM Trans. Math. Software 24 (1998) 437–474.
[11] D. Knapp, V. Barocas, K. Yoo, L. Petzold, R. Tranquillo, Rheology of reconstituted type I collagen gel in confined compression, J. Rheology 41 (1997) 971–993.
[12] W.M. Lioen, J.J.B. de Swart, Test set for initial value problem solvers, Technical Report MAS-R9832, CWI, Amsterdam, 1998.
[13] S. Li, L. Petzold, Design of new DASPK for sensitivity analysis, Technical Report, Department of Computer Science, UCSB, 1999.
[14] S. Li, L. Petzold, W. Zhu, Sensitivity analysis of differential-algebraic equations: a comparison of methods on a special problem, Appl. Numer. Math. 32 (2000) 161–174.

[15] S. Li, L. Petzold, Software and algorithms for sensitivity analysis of large-scale differential-algebraic systems, J. Comput. Appl. Math. 125 (2000) 131–145.
[16] T. Maly, L.R. Petzold, Numerical methods and software for sensitivity analysis of differential-algebraic systems, Appl. Numer. Math. 20 (1997) 57–79.
[17] L. Petzold, P. Lötstedt, Numerical solution of non-linear differential equations with algebraic constraints II, SIAM J. Sci. Statist. Comp. 7 (1986) 720–733.
[18] L.L. Raja, R.J. Kee, R. Serban, L. Petzold, Dynamic optimization of chemically reacting stagnation flows, Proc. Electrochem. Soc. (1998).