Description of DASPKADJOINT: An Adjoint Sensitivity Solver for Differential-Algebraic Equations

Shengtai Li and Linda Petzold [*]

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

Abstract

DASPKADJOINT, for sensitivity computation of differential-algebraic equations (DAEs) by the adjoint method, is introduced and described. Several critical issues for the implementation are addressed. DASPKADJOINT is more efficient than the forward method implemented in DASPK3.0 for applications with a large number of sensitivity parameters and few objective functions. Numerical examples are given to illustrate the simple and effective use of the software.

1 Introduction

In this report we describe new software, called DASPKADJOINT, for the computation of the sensitivities of differential-algebraic equations (DAE) by the adjoint method. Given a DAE depending on parameters,

$$\left\{ \begin{array}{rcl} F(y, \dot{y}, t, p) & = & 0 \\ y(0) & = & y_0(p), \end{array} \right. \tag{1}$$

and a vector of objective functions $G(y, p)$, the sensitivity problem usually takes the form: find $\frac{dG}{dp}$, where p is a vector of parameters. By the chain rule, the sensitivity $\frac{dG}{dp}$ is given by

$$\frac{dG}{dp} = \frac{\partial G}{\partial y}\frac{dy}{dp} + \frac{\partial G}{\partial p}. \tag{2}$$

Let n_y, n_p and n_G be dimensions of y, p and G respectively. The computation of $\frac{dy}{dp}$ in (2) 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

[*]This work was supported by grants: DOE DE-FG03-00ER25430, NSF/FTR CCR 98-96198, NSF/KDI ATM-9873133 and LLNL ISCR 00-15.

the same Jacobian matrices with the original system. This method has been implemented in DASPK3.0 [21].

Some problems require the sensitivities with respect to a large number of parameters. For these problems, particularly if n_y is large, the forward sensitivity approach is intractable. These problems can often be handled more efficiently by the adjoint method [12]. To see how the adjoint method works, consider the problem of finding $\frac{dG(y,p)}{dp}$, where y is the solution of a nonlinear system $H(y,p) = 0$. We have the following relationship

$$\frac{\partial H}{\partial y}\frac{dy}{dp} + \frac{\partial H}{\partial p} = 0.$$

Assuming that $\frac{\partial H}{\partial y}$ is boundedly invertible, the sensitivity $\frac{dG}{dp}$ is given by

$$\frac{dG}{dp} = -\frac{\partial G}{\partial y}\left(\frac{\partial H}{\partial y}\right)^{-1}\frac{\partial H}{\partial p} + \frac{\partial G}{\partial p}. \tag{3}$$

Rather than computing $\frac{dy}{dp}$ directly, which could be very expensive if n_y and n_p are large, we solve for the adjoint variable λ, where $\lambda^* = -\frac{\partial G}{\partial y}\left(\frac{\partial H}{\partial y}\right)^{-1}$, which gives

$$\lambda^*\frac{\partial H}{\partial y} = -\frac{\partial G}{\partial y}^* . \tag{4}$$

Thus if n_G is small (for example $n_G = 1$ for an optimization problem), we can obtain the sensitivity very efficiently by solving a small system.

For the DAE system (1), the adjoint sensitivity method and the corresponding adjoint DAE can be derived similarly. In [10] we derived the adjoint sensitivity system for DAEs of index up to two (Hessenberg) and investigated some of its fundamental properties. In [11] we addressed some of the issues for the numerical solution. In this report, we describe the adjoint DAE solver, giving details about its use and implementation.

This report is organized as follows. In Section 2, we outline the adjoint sensitivity method for DAEs and summarize some of the relevant results from [10] and [11]. In Section 3 we describe how to evaluate the adjoint DAE efficiently by an automatic differentiation tool and how to initialize the adjoint DAE with the help of existing mechanisms in DASPK3.0. Section 5 describes some important considerations for implementation of the adjoint sensitivity method for DAEs and how they are addressed in our software, DASPKADJOINT. Finally, the effectiveness and efficiency of the algorithms and software are demonstrated on several examples in Section 7.

2 The Adjoint DAE System and Sensitivity Calculation

In this section we present the adjoint sensitivity system for DAEs and summarize some of the relevant results concerning initial values, stability and numerical stability from [10].

The adjoint system for the DAE

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

with respect to the derived function $G(y, p)$

$$G(y, p) = \int_0^T g(t, y, p) dt \tag{5}$$

is given by

$$(\lambda^* F_{\dot{y}})' - \lambda^* F_y = -g_y, \tag{6}$$

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 (6) are taken to be $\lambda^* F_{\dot{y}}|_{t=T} = 0$, and the sensitivities of $G(y, p)$ with respect to the parameters p are given by

$$\frac{dG}{dp} = \int_0^T (g_p - \lambda^* F_p)\, dt + (\lambda^* F_{\dot{y}})|_{t=0} y_{0p}. \tag{7}$$

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 (7)[10].

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

$$(\lambda_T^* F_{\dot{y}})' - \lambda_T^* F_y = 0, \tag{8}$$

where λ_T denotes $\frac{\partial \lambda}{\partial T}$. For index-0 and index-1 DAE systems, the initial conditions $\lambda_T(T)$ for (8) satisfy $(\lambda_T^* F_{\dot{y}})|_{t=T} = [g_y - \lambda^* F_y]\,|_{t=T}$. 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 $g(y, T, p)$ with respect to the parameters p are given for index-0 and index-1 DAE systems by

$$\frac{dg}{dp} = (g_p - \lambda^* F_p)|_{t=T} - \int_0^T (\lambda_T^* F_p) + (\lambda_T^* F_{\dot{y}})|_{t=0} (y_0)_p. \tag{9}$$

Note that the values of both λ at $t = T$ and λ_T at $t = 0$ are required in (9). 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 (9).

We focus on the adjoint system with respect to the scalar objective function $g(y, T, p)$ throughout this paper. If the objective function is of the integral form $G(y, p)$ (5), it can be computed easily by appending a *quadrature variable* (and corresponding equation), 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 y_{N+1} and equation

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

Then $G = y_{N+1}(y, T, p)$. In this way, we can transform any objective function in the integral form (5) into the scalar form $g(y, T, p)$. The quadrature variables can be calculated very efficiently [21] by a staggered method in DASPK3.0; they do not enter into the Jacobian matrix.

From [10] 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,

$$\begin{aligned} \dot{\bar{\lambda}} - F_y^* \lambda &= 0, \\ \bar{\lambda} - F_{\dot{y}}^* \lambda &= 0. \end{aligned} \tag{10}$$

It was shown in [10] that the system (10) with respect to $\bar{\lambda}$ preserves the stability of the original 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 (8). This yields

$$\dot{\lambda}_T^* F_{\dot{y}}(T - \tau, y) + \lambda_T^* \left(F_y(T - \tau, y) - \frac{dF_{\dot{y}}}{dt}(T - \tau, y) \right) = 0, \tag{11}$$

where $\dot{\lambda}_T^* = d\lambda_T^*/d\tau$, $\dot{y} = dy/dt$, and $\frac{dF_{\dot{y}}}{dt}$ is the total derivative of $F_{\dot{y}}$ with respect to t. Another advantage of using (11) instead of (8) is that the Jacobian for (11) is exactly the transpose of that for the forward state computation. If the DAE system is linear with respect to $\dot{y}$, then for any constant vector v, the product $\frac{dF_{\dot{y}}}{dt}$ can be evaluated by

$$v \frac{dF_{\dot{y}}}{dt} = vF_{\dot{y}t} + (vF_{\dot{y}})_y \dot{y}. \tag{12}$$

If the DAE system is nonlinear with respect to $\dot{y}$, an additional variable can be added to make the DAE linear with respect to $\dot{y}$.

For many applications, $F_{\dot{y}}$ is constant and equation (11) becomes

$$\dot{\lambda}_T^* F_{\dot{y}} + \lambda_T^* F_y = 0. \tag{13}$$

In these cases, we do not need to evaluate the term $\frac{dF_{\dot{y}}}{dt}$. If $F_{\dot{y}}$ is time-varying, the augmented adjoint DAE system (10), which becomes

$$\begin{aligned} \dot{\bar{\lambda}} + F_y^* \lambda &= 0, \\ \bar{\lambda} - F_{\dot{y}}^* \lambda &= 0, \end{aligned} \tag{14}$$

under the time-reversing transformation, is used to preserve the stability. Although the evaluation of (12) may not occur during the integration, it still has to be done during the initialization.

Equations (13) and (14) involve matrix-vector products from the left side (referred to as vector-matrix products in the following). Although a matrix-vector product $F_y v$ can be approximated via a directional derivative finite difference method, it is difficult to evaluate the vector-matrix product vF_y directly via a finite difference method. The vector-matrix product vF_y can be written as a gradient of the function $vF(y)$ with respect to y. If a finite-difference method is used to calculate the gradient of $vF(y)$, n_y evaluations of $vF(y)$ are required. However, the cheap gradient theorem [15] tells us that the gradient of any nonlinear function can be calculated for less than the cost of 5 function evaluations by automatic differentiation (AD). Therefore, AD is necessary for full computational efficiency. A forward mode AD tool cannot compute the vector-matrix products without evaluation of the full Jacobian. It has been shown [14] 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 DASPKADJOINT, we use the AD tool TAMC [14] to calculate the vector-matrix products. In contrast to the forward mode, it is not possible with the backward mode to evaluate (11) by just one automatic differentiation. Therefore we need to evaluate each vector-matrix product separately. In order to reduce the computational effort for a range of problems, we have three options to evaluate the adjoint DAE, corresponding to three different formulations of the original DAE:

- If the DAE is of explicit ODE form, say $\dot{y} = f(t, y)$, then the adjoint DAE is $\dot{\lambda}^* = \lambda^* f_y$, and only one application of AD is necessary;

- If the DAE is of the general form $F(t, y, \dot{y}) = 0$ and $F_{\dot{y}}$ is constant, then the adjoint DAE is $\dot{\lambda}^* F_{\dot{y}} + \lambda^* F_y = 0$, and two applications of AD for two different vector-matrix products are needed.

- If the DAE is of the general form $F(t, y, \dot{y}) = 0$ and $F_{\dot{y}}$ depends on t and/or y, and consistent initialization is required, then the adjoint DAE is of the form (11), where $\lambda^* F_y$ can be evaluated by AD with reverse mode and $\dot{\lambda}^* F_{\dot{y}} - \lambda \frac{dF_{\dot{y}}}{dt}$ can be evaluated by AD with a combination of forward and reverse modes. First a routine is generated by an application of AD with reverse mode to calculate the vector-matrix product $\lambda F_{\dot{y}}$. Then a routine to compute the total derivative of $\lambda F_{\dot{y}}$ with respect to time is generated by an application of AD with forward mode.

If the adjoint DAE can be readily obtained, there is an option in DASPKADJOINT for the user to input the adjoint DAE directly without making use of operation.

ADIFOR [3] is another AD tool. ADIFOR2.0 includes only the forward mode (ADIFOR3.0 has the backward mode but is not released to the public yet). If the SparseLinC option is used, ADIFOR can evaluate the nonzero 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 describe the initialization procedures for the adjoint DAE with respect to the scalar objective function $g(y, T, p)$. From Section 2, we know that the adjoint variables must satisfy initial conditions at time $t = T$. Thus, in order to reduce the computational effort for different forms of DAE system, we have three options in DASPKADJOINT.

4.1 Index-0 and semi-explicit index-one case

The simplest case is the standard ODE form, where $\lambda_T = g_y$ can be directly computed. The second case is when the DAE is of regular implicit form (nonsingular mass matrix $F_{\dot{y}}$), or of the following index-1 form:

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

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

$$\begin{aligned} M(t, y^d)^*\dot{\lambda}^d &= (f_{y^d} + dM/dt - d(M\dot{y}^d)/dy^d)^*\lambda^d + h^*_{y^d}\lambda^a + g_{y^d}, \\ 0 &= f^*_{y^a}\lambda^d + h^*_{y^a}\lambda^a + g_{y^a}, \end{aligned}$$

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 $\lambda^d_T(T) = \dot{\lambda}^d(T)$ and initialize the adjoint DAE

$$\begin{aligned} M(t, y^d)^*\dot{\lambda}^d_T &= (f_{y^d} + dM/dt - d(M\dot{y}^d)/dy^d)^*\lambda^d_T + h^*_{y^d}\lambda^a_T, \\ 0 &= f^*_{y^a}\lambda^d_T + h^*_{y^a}\lambda^a_T, \end{aligned}$$

for the objective function $g(y, T, p)$. The differential variables $\lambda^d_T(T)$ are fixed during the second step. Each initialization can be done easily in DASPK3.0 [22].

For fully-implicit index-1 DAE systems

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

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

$$\begin{aligned} A^*\dot{\lambda} + B^*\lambda &= g_y, \\ A^*\lambda &= 0, \end{aligned}$$

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

4.2 Hessenberg index-two DAE systems

For Hessenberg index-2 DAE systems,

$$\begin{aligned} \dot{y}^d &= f(t, y^d, y^a, p), \\ 0 &= h(t, y^d, p), \end{aligned} \tag{15}$$

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

$$\begin{aligned} \dot{\lambda}_T^{d*} &= -\lambda_T^{d*} f_{y^d} - \lambda_T^{a*} h_{y^d}, \\ 0 &= \lambda_T^{d*} f_{y^a}. \end{aligned}$$

The initial values for the adjoint variable λ_T^d satisfy ([10])

$$\lambda_T^d(T) = P^*(g_{y^d}^* + f_{y^d}^* \lambda^d - \dot{h}_{y^d}^* (f_{y^a}^* h_{y^d}^*)^{-1} g_{y^a}^*)|_{t=T}, \tag{16}$$

or

$$\lambda_T^{d*}(T) = (g_{y^d} + \lambda^{d*} f_{y^d} - g_{y^a}(h_{y^d} f_{y^a})^{-1} \dot{h}_{y^d}) P|_{t=T}, \tag{17}$$

where $\lambda^{d*} = -g_{y^a}(h_{y^d} f_{y^a})^{-1} h_{y^d}$ is the adjoint variable for the objective function $G = \int_0^T g(t, y, p)dt$, $P = I - f_y(h_y f_y)^{-1} h_y$ is a projection matrix for the original index-2 system, and $\dot{h}_{y^d}$ is the total derivative of h_{y^d} with respect to t, $\dot{h}_{y^d} = h_{y^d t} + (h_{y^d} \dot{y}^d)_{y^d}$. Note that f_{y^d}, f_{y^a}, h_{y^d} and $\dot{h}_{y^d}$ 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 $\lambda^{d*}(T) = -g_{y^a}(h_{y^d} f_{y^a})^{-1} h_{y^d}|_{t=T}$ by solving the following initialization problem

$$\begin{aligned} \dot{\lambda}_1^{d*} &= \lambda_1^{d*} f_{y^d} + \lambda_1^{a*} h_{y^d}, \\ 0 &= \dot{\lambda}_1^{d*} f_{y^a} - \lambda_1^{d*} \dot{f}_{y^a} - g_{y^a}, \end{aligned} \tag{18}$$

for $\dot{\lambda}_1^d(T)$ and $\lambda_1^a(T)$ with $\lambda_1^d(T) = 0$ fixed. In (18), $\dot{f}_{y^a}$ represents the total derivative of f_{y^a} with respect to t. Note that $\lambda_1^{a*}(T) = g_{y^a}(h_{y^d} f_{y^a})^{-1}|_{t=T}$, and $\lambda^{d*}(T) = -\dot{\lambda}_1^{d*}(T) = -g_{y^a}(h_{y^d} f_{y^a})^{-1} h_{y^d}|_{t=T}$. If $g_{y^a} = 0$, then $\lambda^{d*}(T) = 0$.

In step 2, we calculate

$$v_1 = g_{y^d} + \lambda^{d*} f_{y^d} - g_{y^a}(h_{y^d} f_{y^a})^{-1} \dot{h}_{y^d}|_{t=T} = g_{y^d} + \lambda^{d*} f_{y^d} - \lambda_1^{a*} \dot{h}_{y^d}|_{t=T}. \tag{19}$$

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

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

$$\lambda_T^{d*}(T) = v_1 P|_{t=T} = v_1(I - f_{y^a}(h_{y^d} f_{y^a})^{-1} h_{y^d})|_{t=T}, \tag{20}$$

by initializing the following system

$$\begin{aligned} \dot{\lambda}_2^{d*} &= \lambda_2^{d*} f_{y^d} + \lambda_2^{a*} h_{y^d} + v_1, \\ 0 &= \dot{\lambda}_2^{d*} f_{y^a} - \lambda_2^{d*} \dot{f}_{y^a}, \end{aligned} \tag{21}$$

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

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

$$\begin{aligned} \dot{\lambda}_T^{d*} &= \lambda_T^{d*} f_{y^d} + \lambda_T^{a*} h_{y^d}, \\ 0 &= \dot{\lambda}_T^{d*} f_{y^a} - \lambda_T^{d*} \dot{f}_{y^a}, \end{aligned} \tag{22}$$

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

The DAE systems (18) and (21) have the same format except for the forcing terms. Therefore, we can solve them efficiently using the same algorithm. The second equation of the adjoint DAE system (22) is actually the derivative of the index-2 constraint $\lambda^d f_{y^a}$ 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 [21].

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

$$\begin{aligned} M(t, y^d)\dot{y}^d &= f(t, y^d, y^a), \\ 0 &= h(t, y^d), \end{aligned} \tag{23}$$

where $M(t, y^d)$ is a nonsingular square matrix, the adjoint system is

$$\begin{aligned} M(t, y^d)^* \dot{\lambda}_T^d &= (f_{y^d} + \dot{M} - (M\dot{y}^d)_{y^d})^* \lambda_T^d + h_{y^d}^* \lambda_T^a, \\ 0 &= f_{y^a}^* \lambda_T^d. \end{aligned} \tag{24}$$

The initial values for the adjoint variables λ_T^d satisfy

$$\begin{aligned} \lambda_T^{d*} &= \left(\dot{\lambda}^d - g_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1} \frac{d(h_{y^d} M^{-1})}{dt} \right) P, \\ &= \left(\left(g_{y^d} + \lambda^{d*}(f_{y^d} + \dot{M} - (M\dot{y}^d)_{y^d}) \right) M^{-1} - g_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1} (\dot{h}_{y^d} M^{-1} - h_{y^d} M^{-1} \dot{M} M^{-1}) \right) P \\ &= \left(g_{y^d} + \lambda^{d*}(f_{y^d} - (M\dot{y}^d)_{y^d}) - g_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1} \dot{h}_{y^d} \right) M^{-1} P \\ &= (g_{y^d} - \lambda^{d*} F_{y^d} - \lambda_1^{a*} \dot{h}_{y^d}) M^{-1} P \\ &= v_1 M^{-1} P \end{aligned}$$

at $t = T$, where $\lambda^{d*} = -g_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1} h_{y^d} M^{-1}$, $P = I - f_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1} h_{y^d} M^{-1}$, $F = M\dot{y}^d - f(y^d, y^a, p)$, $\lambda_1^{a*} = g_{y^a}(h_{y^d} M^{-1} f_{y^a})^{-1}$, and $v_1 = g_{y^d} - \lambda^{d*} F_{y^d} - \lambda_1^{a*} \dot{h}_{y^d}$. The above initialization procedures for Hessenberg form can still be used by replacing the adjoint DAE system with (24).

5 Implementation

In this section we outline some of the issues for implementation of the adjoint method for DAE sensitivity analysis. We use our software DASPK3.0 [21] as the backbone DAE solver.

DASPK3.0 already includes a forward method to compute the sensitivity efficiently. It might seem appealing to incorporate the adjoint method into it without introducing another name. However, the adjoint DAE requires the solutions of the state variables before it begins the backward integration. In fact, the main solver of DASPK3.0 is used frequently by the adjoint method. Moreover, the adjoint method needs many arguments and routines that are never used in the integration of the state variable or in the forward sensitivity method. To maintain the compatibility of DASPK3.0 with previous versions of DASPK and DASSL without introducing too many unnecessary complications for the integration of the state equations, we have introduced another program DASPKADJOINT to take care of the adjoint method, and have modified DASPK3.0 only slightly to accommodate its use by DASPKADJOINT.

The implementation of DASPKADJOINT 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 (6). Finally, we solve the adjoint system backward to the initial point, and calculate the sensitivities.

5.1 Checkpointing technique

In the adjoint system (8) and the sensitivity calculation (9), the derivatives F_y, $F_{\dot{y}}$ and F_p may depend on the state variables y, which are the solutions of the original DAEs. Ideally, the adjoint DAE (8) 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.

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 y and $\dot{y}$ 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 y, 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

[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.

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 a 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 for some applications that the number of checkpoints is also too large 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.

5.2 Program structure

The forward integration may be performed twice (if checkpointing is used) but only one initialization is required. Hence, if consistent initialization for the state variables is required, DASPKADJOINT always performs the initialization first without the integration. Then DASPKADJOINT performs the forward integration of the state variables by calling DASPK3.0. We return from DASPK3.0 after every time step to see if a checkpoint has been encountered. At the checkpoint, the necessary information to repeat the integration from the checkpoint is stored. After the forward integration is completed, we calculate the objective function and its gradients with respect to y and p. Then we initialize the adjoint DAE. Several options have been provided for index-0, index-1 and index-2 DAE systems. The next step is the backward integration. In each checkpointing interval, we first recover the data at the checkpoint and then start the forward integration from there. The state variables and their time derivatives at each time step are stored to be accessed later by the backward integration. After the internal forward integration has been completed, we integrate the adjoint system backward to the checkpoint. This process is repeated for each checkpoint until $t = t_0$ is reached. Finally, we calculate the sensitivity based on the adjoint variables at $t = t_0$.

We remark that our program structure can be used for any adjoint DAE solver. It is not limited to the DASPK3.0 and DASPKADJOINT. However the backbone forward DAE solver needs to be modified slightly. In the following subsections, we will describe how to make those changes.

5.3 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 nonlinear system at each time step. When the mass matrix $F_{\dot{y}}$ is constant, the Jacobian for the adjoint DAE is exactly the transpose of the Jacobian for the original DAE. Therefore we can evaluate the Jacobian of the original DAE and transpose it to obtain the Jacobian for the adjoint DAE.

If $F_{\dot{y}}$ depends on either t or y, we solve the augmented adjoint DAE (14). The Jacobian for (14) is

$$\begin{pmatrix} CJ \cdot I & F_y^* \\ I & -F_{\dot{y}}^* \end{pmatrix}$$

where CJ is a scalar coefficient determined by DASPK3.0. If we solve only for λ, then the second column is multiplied by $-CJ$ and added to the first column, which yields the Jacobian for λ only, $CJ \cdot F_{\dot{y}}^* + F_y^*$, 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

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

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 (25) 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 nonlinear.

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

5.4 Krylov iterative method

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

(26) $$u = (\alpha F_{\dot{y}}^* + F_y^*)v.$$

Since we already have a mechanism to evaluate $vF_{\dot{y}}$ and vF_y when we evaluate the adjoint DAE, (26) 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 easily be obtained by transposing the Jacobian first and then doing the incomplete LU factorization.

5.5 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 (14),

$$\begin{aligned} \frac{\bar{\lambda}_{n+1} - \bar{\lambda}_n}{h_{n+1}} + F_y^* \lambda_{n+1} &= 0, \\ \bar{\lambda}_{n+1} - F_{\dot{y}}^* \lambda_{n+1} &= 0. \end{aligned} \tag{27}$$

The true solution to (14) satisfies

$$\begin{aligned} \frac{\bar{\lambda}(t_{n+1}) - \bar{\lambda}(t_n) - \frac{h_{n+1}^2}{2}\bar{\lambda}''(\xi)}{h_{n+1}} + F_y^* \lambda(t_{n+1}) &= 0, \\ \bar{\lambda}(t_{n+1}) - F_{\dot{y}}^* \lambda(t_{n+1}) &= 0. \end{aligned} \tag{28}$$

Subtracting (28) from (27), we obtain

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

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 [24], we can consider removing 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.

5.6 Sensitivity calculation

The integral in equation (9) 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

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

λ_{N+1} is called a quadrature variable in DASPK3.0 and is calculated efficiently [21] without participating in Newton iterations and Jacobian evaluations via a staggered method. The other terms in (9) are easily obtained: $\lambda^* F_p$ and $\lambda_T F_{\dot{y}}$ can be computed by an AD tool with reverse mode.

$(y_0)_p$ is the Jacobian matrix of the initial condition y_0 with respect to the sensitivity parameter p. When p is the initial condition for an ODE (or implicit ODE), $(y_0)_p = I$ is the identity matrix. For an index-1 or index-2 system, $(y_0)_p$ must be consistent with the algebraic constraints and/or any hidden constraints. $(y_0)_p$ can either be input by the user or evaluated by an AD tool. When the number of parameters is large, the matrix $(y_0)_p$ is huge. However, the number of nonzero elements in $(y_0)_p$ may be small. A sparse format to store and compute $(y_0)_p$ is necessary.

For the Hessenberg index-2 DAE system (15), the value of λ^{a*} is required when $h_p \neq 0$, because $\lambda^{a*} h_p$ should be calculated in the sensitivity (9). To solve for λ^{a*} correctly in (19), we need the value of $\frac{dg_{y^a}}{dt}$, which is difficult for the user to provide. However, there is an additional term for the sensitivity (9) if $h_p \neq 0$, which is $\frac{d}{dt}(-g_{y^a}(h_{y^d} f_{y^a})^{-1} h_p)$. The combination of the two terms will be

$$\begin{aligned}
& (\lambda^{d*} f_p + \lambda^{a*} h_p) + \frac{d}{dt}(-g_{y^a}(h_{y^d} f_{y^a})^{-1} h_p) \\
= \; & \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + \left(\lambda^{a*} - \frac{dg_{y^a}}{dt}(h_{y^d} f_{y^a})^{-1} + g_{y^a}(h_{y^d} f_{y^a})^{-1} \left(\dot{h}_{y^d} f_{y^a} + h_{y^d} \dot{f}_{y^a} \right) (h_{y^d} f_{y^a})^{-1} \right) h_p \\
= \; & \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + (\lambda^{a*} h_{y^d} - \dot{\lambda}^{d*} + \lambda_1^{a*} \dot{h}_{y^d}) f_{y^a} (h_{y^d} f_{y^a})^{-1} h_p \\
= \; & \lambda^{d*} f_p - \lambda_1^{a*} \dot{h}_p + \lambda_2^{a*} h_p,
\end{aligned}$$

where $\lambda^{d*} f_{y^a} + g_{y^a} = 0$ is used, and $\lambda_1^{a*} = -g_{y^a}(h_{y^d} f_{y^a})^{-1}$ and $\lambda_2^{a*} = -(g_{y^d} + \lambda^{d*} f_{y^d} - \lambda_1^{a*} \dot{h}_{y^d}) f_{y^a} (h_{y^d} f_{y^a})^{-1}$ can be obtained from equations (18) and (21) respectively. It can be verified that the last equation is also valid for the case of non-Hessenberg form given by (23).

6 Obtaining Numerical Sensitivities with DASPKADJOINT

6.1 Getting started with DASPKADJOINT

DASPKADJOINT is designed to be as easy to use as possible, while providing enough flexibility and control for solving a wide variety of problems. It is extensively documented in the source code. The user interface for DASPKADJOINT is based on the interface for DASPK3.0, with a few changes which are necessary for the adjoint DAE and the sensitivity calculation. Since DASPKADJOINT is designed based on DASPK3.0, we strongly recommend that the user read the documentation for DASPK3.0 first. In this subsection, we outline what a user must do to compute the sensitivities of a vector of objective functions with DASPKADJOINT.

We emphasize that DASPKADJOINT is designed for solving index-zero, index-1 and index-2 DAE systems with nonsingular mass matrix. The DAE system and its initial condition are posed as

$$\begin{aligned} F(t,y,y') &= 0 \\ y(t_0) &= y_0 \\ y'(t_0) &= y'_0, \end{aligned} \tag{29}$$

where F, y and y' are N_y dimensional vectors. As in DASPK3.0, the function F in (29) is defined in a subroutine RES which is written by the user. RES has the same argument list as in DASPK3.0. It takes T and the vectors Y and YPRIME as inputs, and produces an output vector DELTA, where DELTA = F(T,Y,YPRIME). The subroutine has the form

```
SUBROUTINE RES(T,Y,YPRIME,CJ,DELTA,IRES,RPAR,IPAR,SENPAR)
```

The parameter CJ can be ignored or used to scale the algebraic constraints if necessary. IRES is an integer flag. Depending on the value of IRES, RES can be used to define the residuals for different situations. If IRES=0, RES is used to define the residuals for the state equations excluding the quadrature equations; if IRES=2, RES is used to define the residuals of the time derivatives of the index-2 constraints; if IRES=3, RES is used to define the residuals for the quadrature equations. IRES is also used to flag situations where an illegal value of Y or a stop condition has been encountered. The user would set IRES=-1 and return without evaluating the function in those situations. RPAR and IPAR are real and integer vectors respectively, and are at the user's disposal to use for communication purposes. SENPAR is a real parameter array for sensitivity computation. It contains only those sensitivity parameters that appear in RES. Those parameters that appear only in the initial conditions are not stored in SENPAR.

In DASPK3.0, we suggest that the user scale the index-2 constraints by CJ for an index-2 DAE system to reduce the round-off error [6]. The index-2 constraints correspond to the index-2 variables in the adjoint DAE system and the index-2 variables in the original DAE correspond to the index-2 constraints in the adjoint DAE system. Therefore, we must scale the index-2 variables by 1/CJ and scale the index-2 constraints by CJ in the adjoint DAE system. Similarly, the Jacobian should be evaluated in a different way for the index-2 system. The transpose of the original DAE needs scaling of some rows and columns to become the Jacobian of the adjoint DAEs. Specifically, the rows corresponding to the index-2 constraints should be scaled by CJ, and the columns corresponding to the index-2 variables should be scaled by 1/CJ.

Another routine the user must provide is the subroutine that defines the objective function $G = g(t,y,p)$ and/or its partial derivatives with respect to y and p. The user has an option to define the function G only and let DASPKADJOINT compute the gradient with respect to y and p through a finite-difference method or automatic differentiation. This subroutine has the form

SUBROUTINE QRES(T,Y,YPRIME,QSEN,IRES,RPAR,IPAR,SENPAR)

The parameter QSEN is a real array that contains the value of G and its partial derivatives. QSEN has dimension at least $N_g(N_y+1+N_p)$, where N_g is the number of objective functions and N_p is the total number of sensitivity parameters. On return from DASPKADJOINT, QSEN stores the sensitivity information.

To get started, the initial values of T, Y and YPRIME must be given. If they are not consistent, a consistent initialization option has to be chosen, which is specified in the INFO array. The call to DASPKADJOINT is

```
CALL DDASPKADJOINT(
     RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC, PSOL, SENPAR,
     ADRES, NQ, QRES, QSEN, INFOB, RTOLB, ATOLB, NBUF,
     ADJAC, IEOPT, RES_ADP, RES_ADY, G_RES_ADY, G_RES_ADYP,
     ADINIT, K_RES, T_RES, G_RES_ADP)
```

The parameters are described in detail in the documentation of DASPK3.0 and DASPKADJOINT, so we will only discuss a few features here. Many of these features are activated by setting an element of the option vectors INFO and INFOB to a positive number. INFO is for the solution of the state variable Y. INFOB is for the evaluation of the adjoint system and the sensitivity computation. Setting up the INFO array for DASPKADJOINT is almost the same as for DASPK3.0 with the exception of a few changes that are documented in the source code of DASPKADJOINT. INFOB is new for DASPK users.

INFOB(1) is used to set up the error tolerance for the adjoint variables. Since the adjoint DAE uses the numerical solutions of the state variables, the default tolerances for the adjoint variables are double the tolerances that are used for the state variables. The user can input different tolerances in RTOLB and ATOLB by setting INFO(1)=1.

INFOB(2) is for the consistent initialization of the adjoint DAE. Consistent initial conditions for the adjoint DAEs are very easy to obtain for an explicit ODE. However for an implicit ODE or a DAE, especially an index-2 DAE, the initial conditions are complicated. Although the user might want to input the consistent initial conditions for efficiency by setting INFO(2)=3 and providing routine ADINIT

```
SUBROUTINE ADINIT(T,NEQ,NQ,Y,YPRIME,ADY,ADYPRIME,QSEN,
           RPAR,IPAR,SENPAR)
```

we strongly recommend the user let DASPKADJOINT compute the consistent initial conditions by setting INFOB=0,1,2, which corresponds to explicit ODE, index-0 and index-1 DAE, and index-2 DAE respectively.

INFOB(3) is for the evaluation of the adjoint DAE system. The adjoint system can be constructed either by AD tools inside DASPK3.0, or by the user outside DASPK3.0. If AD is chosen, the AD-generated routine must be provided by the user. In our interface, we have a fixed argument list for the AD-generated routine. However, the AD-generated routines by some AD-tools, for example TAMC, may have different argument lists from what we require. Therefore, the user needs to modify the argument list of the AD-generated routine so that the arguments match correctly with the interface. A simple way to do that is to write a wrapper for the AD-generated routine. ADRES, RES_ADP, RES_ADY, G_RES_ADY, G_RES_ADYP, and G_RES_ADP are the AD-generated routines. However, not every routine is required for a specific problem. Depending on what option you have used for the evaluation, some routines may be ignored or declared as dummys.

- ADRES is required when $F_{\dot{y}}$ is not the identity. It calculates the vector-matrix product $vF_{\dot{y}}$.
- RES_ADP is required when the sensitivity parameters appear in the RES routine explicitly, i.e., the dimension of SENPAR is greater than zero. It calculates the vector-matrix product vF_p.
- RES_ADY is required except when the user chooses to evaluate the adjoint system outside DASPK3.0. It calculates the vector-matrix vF_y.
- G_RES_ADY is required for the initialization of an index-2 DAE system. It computes the total time derivative of the vector-matrix product $\lambda^* F_y$.
- G_RES_ADYP is required for the initialization of the adjoint system when $F_{\dot{y}}$ depends on t and/or y. It computes the total time derivatives of the vector-matrix product $\lambda^* F_{y'}$.
- G_RES_ADP is required for the sensitivity calculation of an index-2 DAE system where the sensitivity parameters appear explicitly in the RES. It calculates the total time derivatives of the vector-matrix product $\lambda^* F_p$.

If the user chooses to evaluate the adjoint DAE system outside DASPK3.0, ADRES should be provided and has the following argument list

```
SUBROUTINE ADRES(T,ADY,ADYPRIME,CJ,DELTA,IRES,RPAR,IPAR,SENPAR,Y)
```

where Y represents the state variables, and ADY and ADYPRIME represent the adjoint variables and their time derivatives.

An index-2 DAE system requires special scaling in the constraints and the Jacobian. If the system is of index-2, INFOB(5) must be set to 1 and an IEOPT array must be provided to indicate which equations are index-2 constraints and which variables are index-2 variables. IEOPT should be set as

- IEOPT(i) = -1, if the ith equation is an index-1 constraint;
- IEOPT(i) = -2, if the ith equation is an index-2 constraint;
- IEOPT(i) = 1, if the ith equation is a differential equation.
- IEOPT(i+NEQ)=1,2 or 3 if Y(I) is a differential variable;

- IEOPT(i+NEQ) = -1, if Y(I) is an index-1 algebraic variable;
- IEOPT(i+NEQ) = -2, if Y(I) is an index-2 algebraic variable;

The sensitivity calculation requires the value of $\lambda^* F_{\dot{y}} y_{0p}$ if there are some sensitivity parameters in the initial condition y_0. Usually the initial values of the sensitivities are very simple, and y_{0p} contains a few nonzero elements. Although DASPKADJOINT has an option to compute the sensitivities, it requires the user to input the dense form of y_{0p}, which might be huge if both N_y and N_p are very large. Hence the default option inside DASPKADJOINT is to calculate $\lambda^* F_{\dot{y}}$ only and store it in QSEN. If the user still wants DASPKADJOINT to compute $\lambda^* F_{\dot{y}} y_{0p}$, then set INFOB(6)=1 and provide consistent initial values of y_{0p} in the Y array

If the checkpointing technique is used, the solution of the state variables will need two runs for the adjoint sensitivity method. In the first run, the information at the checkpoints is stored. In the second run, which begins at each checkpoint, the solution of the state variables is stored. These two runs can be reduced to one if the allocated storage is large enough to store the state variables and their time derivatives for every step. Usually, a roughly estimate of the number of time steps needed for the state integration is readily available from previous simulations. Then DASPKADJOINT can make a decision prior to the integration whether the checkpointing technique is needed or not. If the number of time steps for the state integration is known or can be estimated, set INFOB(7) to the upper bound on the number of time steps. DASPKADJOINT will check this number against the size of the allocated storage to see if one run of the state system is enough. If the user does not have an upper bound on the number of time step needed for the forward integration in advance, then set INFOB(7)=0 and DASPKADJOINT will use the checkpointing technique.

If the DAE system is linear with respect to y and $\dot{y}$, then the adjoint system is independent of the state variables y and $\dot{y}$. Hence there is no need to allocate space to store the state variables. Only one run of the forward integration is enough. This option can be chosen by specifying INFOB(3)>3.

Since our goal is to calculate the sensitivities instead of the adjoint variables, we do not insert the adjoint variables and their time derivatives in the argument list of DASPKADJOINT. Instead they use some space in the RWORK array. The RWORK array not only serves as a real work array for the forward state integration, but also server as a real work array for the backward integration. It also serves as the buffer to communicate between the forward and backward integration. The size of RWORK is important for the efficiency of our computation. It should be set as large as possible, because a larger RWORK array means a larger buffer, which then contains more data that can be accessed without resort to a disk file that can slow down the computation.

Finally, we have a few comments on the output of DASPKADJOINT. Unlike DASPK, DASPKADJOINT does not output any sensible sensitivities at intermediate times between T0 and TOUT. If the integration process was interrupted before the task was completed, the user must check the IDID parameter for error messages. Otherwise, on a normal return, the Y array contains the computed solution for the state variables at TOUT. QSEN contains the values of the objective functions and the sensitivity information. Depending on the option the user sets in INFOB(6), the user may need to calculate the sensitivity after calling DASPKADJOINT. Performance data for the backward integration is stored in the

IWORK and RWORK arrays and can be accessed by the user (see the documentation of DASPKADJOINT).

6.2 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. 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.

7 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. The integration methods used are the direct method (D) and Krylov method (K). The sensitivity methods are the forward sensitivity method (F) from DASPK3.0 [21] 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 computations were performed on a Linux machine with Pentium III 450MHZ CPU.

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

7.1 ODE case

7.1.1 2-D Heat equation

We first consider the heat equation

$$u_t = p_1 u_{xx} + p_2 u_{yy}, \tag{30}$$

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 $u_t = 0$. 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 [21]. 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 RTOL = ATOL = 1.0e-5. The size of the buffer was set to such a number that it allows 9 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 memory restrictions, we used only 20 sensitivity parameters (including p_1 and p_2) in the forward sensitivity method. For comparison, we used two objective functions:

$$\begin{aligned} g_1 &= \sum_1^{NEQ} u_i^2, \\ g_2 &= \int_0^T \sum_1^{NEQ} u_i \, dt, \end{aligned}$$

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

Figure 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.

ETH	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

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

7.1.2 Index-0 DAES with state-dependent mass matrix

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

$$\begin{pmatrix} y_1 & y_2 \\ -y_2 & y_1 \end{pmatrix} \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -(y_1^2 + y_2^2) \end{pmatrix}, \tag{31}$$

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

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

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

ETH	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 $y_2 = \cos(t)$, $\dot{\bar{\lambda}} = 0$. 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.

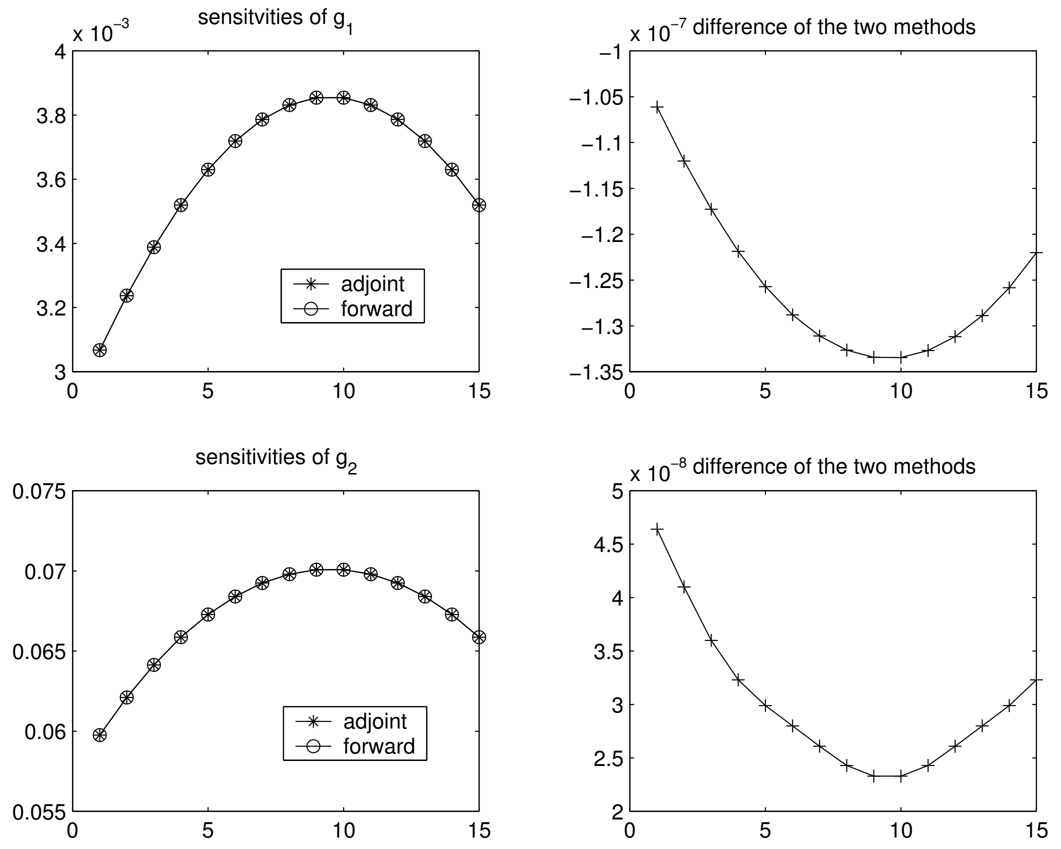

Figure 1: Sensitivities of the objective functions with respect to the initial conditions for the heat equation example. The x-axis represents the order number of the points.

7.2 Index-1 DAE case

7.2.1 2-D food web problem

We consider a multi-species food web [8], in which mutual competition and/or predator-prey relationships in the spatial domain are simulated. Specifically, the model equations for the concentration vector $c = (c^1, c^2)^T$ are

$$\begin{aligned} c^1_t &= f_1(x,y,t,c) + d_1(c^1_{xx} + c^1_{yy}), \\ 0 &= f_2(x,y,t,c) + d_2(c^2_{xx} + c^2_{yy}), \end{aligned}$$

with

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

The coefficients a_{ij}, b_i, d_i are

$$\begin{aligned} &a_{ii} = -1, \quad i = 1,2; \\ &a_{12} = -0.5 \cdot 10^{-6}, \quad a_{21} = 10^4; \\ &a_{ij} = e, \quad i > p \text{ and } j < p; \\ &b_1 = 1 + \alpha xy + \beta \sin(4\pi x)\sin(4\pi y)) = -b_2, \\ &d_1 = 1, \quad d_2 = 0.05. \end{aligned}$$

Error control	Recalculate ρ	NSTP	NRES	NJAC	NETF	NNL	$Err_{\max}$
$\bar{\lambda}$	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

Table 2: Results for example (31). $Err_{\max}$ 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 nonlinear iterations. There are no convergence test failures in any of the tests.

The domain is the unit square $0 \le x, y \le 1$. 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 NEQ $= 2M^2 = 800$. The DASPK3.0 tolerances used were RTOL = 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

$$\begin{aligned} c_0^1 &= 10 + (16x(1-x)y(1-y))^2, \\ c_0^2 &= 100, \end{aligned}$$

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 RTOL = 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

$$\begin{aligned} g_1 &= \sum_1^{NEQ} u_i^2, \\ g_2 &= g_2. \end{aligned}$$

Table 3 gives the results for 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.

ETH	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

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

7.2.2 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

$$\begin{pmatrix} y_2 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} -y_2(y_2-1) \\ y_2 - y_1 - 1 \end{pmatrix} \tag{32}$$

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 ATOL = 10^{-9}, RTOL = 10^{-7}. The results are

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

7.3 Index-2 DAE case

We consider an index-2 DAE from mechanics. This problem is selected from the set of initial value test problems [19]. It is of the form

$$\frac{dy}{dt} = f(y), \quad y(0) = y_0, \quad y'(0) = y_0',$$

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 [19]. 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

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

where

$$\begin{array}{rclcrcll}
\beta_j & = & \alpha_1 \pi & \text{and } \omega_j & = & \frac{2j}{3}\pi + \alpha_2 \pi & \text{for } j = 1, ..., 3, \\
\beta_j & = & \alpha_3 \pi & \text{and } \omega_j & = & \frac{2(j-3)}{7}\pi + \alpha_4 \pi & \text{for } j = 4, ..., 10, \\
\beta_j & = & \alpha_5 \pi & \text{and } \omega_j & = & \frac{2(j-10)}{6}\pi + \alpha_6 \pi & \text{for } j = 11, ..., 16, \\
\beta_j & = & \alpha_7 \pi & \text{and } \omega_j & = & \frac{2(j-17)}{4}\pi + \alpha_8 \pi & \text{for } j = 11, ..., 16.
\end{array}$$

For the remainder of the initial conditions, the reader is referred to [19]. The vector α_i contains the sensitivity parameters, with nominal values

$$\begin{array}{llll}
\alpha_1 = \frac{3}{8}, & \alpha_2 = \frac{1}{13}, & \alpha_3 = \frac{1}{8}, & \alpha_4 = \frac{1}{29}, \\
\alpha_5 = -\frac{2}{15}, & \alpha_6 = \frac{1}{7}, & \alpha_7 = -\frac{3}{10}, & \alpha_4 = \frac{1}{17}.
\end{array}$$

The origin of the problem is to compute the elliptic Fekete points. For any configuration $x := (x_1, x_2, ..., x_N)^T$ in a unit sphere in $\Re^3$, the points $\hat{x}_1, \hat{x}_2, ..., \hat{x}_N$ 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 [19]. 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 8 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 RTOL = ATOL = 10^{-4}. Table 4 gives the results of the adjoint and forward methods. Note that the

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

Table 4: Results for Fekete problem.

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.

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

References

[1] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, 1998.

[2] U. M. Ascher and L. R. Petzold, *Stability of computational methods for constrained dynamics systems*, SIAM J. Sci. Comput. 14, 1 (1993).

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

[4] K. Balla and R. März, *Transfer of boundary conditions for DAEs of index 1*, SIAM J. Numer. Anal. 33, 6 (1996), 2318-2332.

[5] K. Balla and R. März, *Linear differential algebraic equations of index 1 and their adjoint equations*, Results in Mathematics 37 (2000), 13-35.

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

[7] P. N. Brown, A. C. Hindmarsh and L. R. Petzold, *Using Krylov methods in the solution of large-scale differential-algebraic systems*, SIAM J. Sci. Comput. (1994), 1467-1488.

[8] P. N. Brown, A. C. Hindmarsh and L. R. Petzold, *Consistent initial condition calculation for differential-algebraic systems*, SIAM J. Sci. Comput. 19, 5 (1998), 1495-1512.

[9] M. Caracotsios and W. E. Stewart, *Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations*, Computers and Chemical Engineering, 9:4 (1985) 359-365.

[10] Y. Cao, S. Li, L. Petzold and R. Serban, *Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution*, submitted.

[11] Y. Cao, S. Li and L. Petzold, *Adjoint sensitivity analysis for differential-algebraic equations: problem formulation and numerical software*, submitted.

[12] R. M. Errico, *What is an adjoint model?*, Bulletin of the American Meteorological Society 78 (1997), 2577-2591.

[13] W. F. Feehery, J. E. Tolsma and P. I. Barton, *Efficient sensitivity analysis of large-scale differential-algebraic systems*, Applied Numerical athematics, 25 (1997) 41-54.

[14] R. Giering and T. Kaminski, *Recipes for adjoint code construction*, ACM Trans. Math. Software 24 (1998), 437-474.

[15] A. Griewank, *Some bounds on the complexity of gradients, Jacobians, and Hessians,* in Complexity in Nonlinear Optimization, P. Pardolos, ed. World Scientific Publishers, 1993, 131-167.

[16] E. Hairer, S.P. Norsett and G. Wanner, *Solving Ordinary Differential Equations I, Non-stiff problems*, Springer-Verlag, 1987.

[17] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Springer-Verlag, 1991.

[18] D. Knapp, V. Barocas, K. Yoo, L. Petzold and R. Tranquillo, *Rheology of reconstituted type I collagen gel in confined compression*, J. Rheology 41 (1997), 971-993.

[19] W. M. Lioen, J.J.B. de Swart, *Test Set for Initial Value Problem Solvers*, Technical Report MAS-R9832, CWI, Amsterdam, 1998.

[20] S. Li, L. Petzold and W. Zhu, *Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem*, Applied Numerical Mathematics 32 (2000), 161-174.

[21] S. Li and L. Petzold, *Software and algorithms for sensitivity analysis of large-scale differential-algebraic systems*, to appear, J. Comp. and Appl. Math.

[22] S. Li and L. Petzold, *Design of New DASPK for Sensitivity Analysis*, Technical Report, Dept. of Computer Science, UCSB, 1999.

[23] T. Maly and L. R. Petzold, *Numerical methods and software for sensitivity analysis of differential-algebraic systems*, Applied Numerical Mathematics, 20 (1997), 57-79.

[24] L. Petzold and P. Lötstedt, *Numerical solution of nonlinear differential equations with algebraic constraints II,* SIAM J. Sci. Stat. Comput. 7(1986) 720-733.

[25] L. L. Raja, R. J. Kee, R. Serban and L. Petzold, *Dynamic optimization of chemically reacting stagnation flows*, Proc. Electrochemical Society, 1998.