

## CHEAP SECOND ORDER DIRECTIONAL DERIVATIVES OF STIFF ODE EMBEDDED FUNCTIONALS\*

DERYA B. ÖZYURT<sup>†</sup> AND PAUL I. BARTON<sup>†</sup>

**Abstract.** A second order adjoint method is described for calculating directional derivatives of stiff ODE embedded functionals. The derivation of the general directional second order adjoint equations for point- and integral-form functionals is presented. A numerical procedure for calculating these directional derivatives that is relatively insensitive to the number of parameters is described and showcased. By combining automatic differentiation (AD) to obtain the adjoint and sensitivity equations with the staggered corrector method to solve the sensitivity systems, we achieve computational costs noticeably lower than directional finite differences based on a first order adjoint code.

**Key words.** second order adjoint method, Hessian-vector products, staggered corrector method, BDF method

**AMS subject classifications.** 65L10, 65L99, 65L12, 65D30

**DOI.** 10.1137/030601582

**1. Introduction.** The response of a system model to variations in its parameters reveals important information that can be utilized in various engineering and scientific applications. Error analysis of estimated parameters, model discrimination, selection and reduction, computation of derivative information in dynamic optimization, and experimental design all benefit from this sensitivity information.

Sensitivity analysis of a system described by parameter-dependent ODEs has two aims. The first is computing  $\frac{\partial x}{\partial p_i}|_{t,p}$  for all time  $t$  in some interval  $[t_0, t_f]$ , for the real valued parameters  $p_i$ ,  $i = 1, \dots, n_p$ . This can be formulated as follows:

$$(1.1) \quad \dot{x} + F(t, x, p) = 0, \quad x(t_0) = x_0(p),$$

$$(1.2) \quad \frac{\partial \dot{x}}{\partial p_i} + \frac{\partial F}{\partial x} \frac{\partial x}{\partial p_i} + \frac{\partial F}{\partial p_i} = 0, \quad \frac{\partial x(t_0)}{\partial p_i} = \frac{\partial x_0}{\partial p_i}, \quad i = 1, \dots, n_p,$$

where  $x(t, p) \in \mathbb{R}^{n_x}$ . The equations used for sensitivity analysis of ODEs are linear and are constructed via differentiation of the original equations. Recent work has shown that first order forward sensitivities can be calculated reliably, accurately, and efficiently for systems described by stiff ODEs and differential-algebraic equations (DAEs) [11, 13, 15, 5, 16].

The second goal is finding the sensitivity of a scalar valued functional  $G(p)$  of the solution with respect to a large number of parameters. The derivative of the integral-form functional

$$(1.3) \quad G(p) = \int_{t_0}^{t_f} g(t, x(t, p), p) dt$$

---

\*Received by the editors October 31, 2003; accepted for publication (in revised form) July 9, 2004; published electronically April 29, 2005. This work was supported by the National Science Foundation under grant OCE-0205590.

<http://www.siam.org/journals/sisc/26-5/60158.html>

<sup>†</sup>Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139-4307 (derya@mit.edu, pib@mit.edu).

with respect to the parameters can be calculated by introducing a Lagrange multiplier  $\lambda$  to form an augmented objective function

$$(1.4) \quad I(\lambda, p) = G(p) - \int_{t_0}^{t_f} \lambda^T (\dot{x} + F(t, x, p)) dt.$$

Then the sensitivity of  $G$  with respect to  $p$  is

$$(1.5) \quad \frac{\partial G}{\partial p} = \frac{\partial I}{\partial p} = \int_{t_0}^{t_f} (g_p + g_x x_p) dt - \int_{t_0}^{t_f} \lambda^T (F_p + F_x x_p + \dot{x}_p) dt.$$

Hence

$$(1.6) \quad \begin{aligned} \frac{\partial G}{\partial p} &= \int_{t_0}^{t_f} (g_p + g_x x_p) dt - \int_{t_0}^{t_f} (\lambda^T F_p + \lambda^T F_x x_p - \dot{\lambda}^T x_p) dt - (\lambda^T x_p)|_{t_0}^{t_f} \\ &= \int_{t_0}^{t_f} (g_p - \lambda^T F_p) dt - \int_{t_0}^{t_f} [-g_x + \lambda^T F_x - \dot{\lambda}^T] x_p dt - (\lambda^T x_p)|_{t_0}^{t_f}. \end{aligned}$$

By defining the adjoint system as

$$(1.7) \quad \begin{aligned} \dot{\lambda}^T - \lambda^T F_x &= -g_x, \\ \lambda^T(t_f) &= 0, \end{aligned}$$

our derivative becomes

$$(1.8) \quad \frac{\partial G}{\partial p} = \int_{t_0}^{t_f} (g_p - \lambda^T F_p) dt + (\lambda^T x_p)|_{t=t_0}.$$

If we are interested in the sensitivity of a scalar function  $g(x(t_f, p), p)$  at the final time point, i.e.,  $\frac{\partial g}{\partial p}|_{t_f, p}$ , it can be obtained from the above result using the identity  $\frac{\partial g}{\partial p}|_{t_f, p} \equiv \frac{d}{dt_f} \frac{\partial G}{\partial p}|_p$ . This reverse (adjoint) sensitivity analysis is described and implemented for DAEs in [4, 3].

The adjoint method is preferred when there is a single functional and many parameters because only one adjoint system (1.7) is solved compared with  $n_p$  sensitivity systems (1.2). However, to the authors' knowledge, all current implementations of the quadrature calculation (1.8) scale with  $n_p$ . This scaling becomes noticeable when  $n_p$  is large and the solution of (1.1) and (1.7) is relatively inexpensive (see section 4.1). Thus, it is more correct to state that the adjoint method is relatively insensitive to  $n_p$  as compared with the forward sensitivity method. Likewise, the forward sensitivity method is relatively insensitive to the number of functionals whose gradients are required as compared with the adjoint method, which requires a new adjoint system (1.7) for each functional.

Under some circumstances these first order sensitivity analyses can be insufficient. For highly nonlinear systems or when parameter-parameter interactions are important, second order information can prove to be indispensable. Moreover, second order derivatives are required if the sensitivity of system performance to variations in some parameter has to be bounded or minimized [10]. Finally, accurate second order information can improve the performance of many optimization algorithms.

To obtain second order forward sensitivities,  $\frac{\partial^2 x}{\partial p^2}$ , the following  $n_p \times n_p$  systems of linear ODEs have to be solved along with the state (1.1) and first order forward sensitivity (1.2) equations [20] (see Appendix A for notation):

$$\begin{aligned}
 \frac{\partial^2 \dot{x}}{\partial p^2} + [F_x \otimes I_{n_p}] \frac{\partial^2 x}{\partial p^2} + \left[ I_{n_x} \otimes \left( \frac{\partial x}{\partial p} \right)^T \right] \left[ F_{xx} \frac{\partial x}{\partial p} + F_{xp} \right] + \left[ F_{px} \frac{\partial x}{\partial p} + F_{pp} \right] &= 0, \\
 \frac{\partial^2 x(t_0)}{\partial p^2} &= \frac{\partial^2 x_0}{\partial p^2}.
 \end{aligned}
 \tag{1.9}$$

After obtaining the second order sensitivities with respect to the state variables, one can attempt to calculate the second order sensitivities of a functional with respect to a large number of parameters  $\frac{\partial^2 g}{\partial p^2}$  or more importantly  $\frac{\partial^2 g}{\partial p^2} u$ , where  $u$  is a constant direction [2]. However, this method involves solving a number of systems scaling with  $n_p^2$  ( $n_p$  for the directional case). Therefore this route of calculating the second order information is not very attractive when the number of parameters is large.

Similar to the first order case, the second order adjoint method is a way to evaluate derivatives of a functional directly without first calculating the forward second order sensitivities and additional matrix-vector computations.

Second order adjoint analysis has been used to assess the design sensitivity of mechanical system dynamics [10]. It is also employed in meteorological data assimilation problems to calculate the exact gradient and Hessian-vector products with respect to the initial conditions of the state variables [12, 22, 21]. In the former application the second order adjoint system consists of the state equations and three sets of adjoint equations which are solved analytically for two simple examples. In the latter application a discretized version of the code describing the state equations is differentiated to obtain first order forward sensitivity and first and second order adjoint codes.

Directional second order derivatives are often estimated using directional finite differences based on a first order adjoint code,

$$\frac{\partial^2 G}{\partial p^2} \Big|_{p=p^*} u \approx \frac{\nabla G(p^* + \varepsilon u) - \nabla G(p^*)}{\varepsilon},
 \tag{1.10}$$

which requires two state and adjoint integrations, one at  $p^*$  and one at  $p^* + \varepsilon u$ . The cheap gradient result of automatic differentiation (AD) [9] states that vector-matrix products can be evaluated for less than the cost of four function evaluations (in this case a state integration (1.1)). Similarly, according to the cheap Hessian result [9], a directional second order derivative can be calculated for less than the cost of ten function evaluations. Thus, one would not expect a directional second order adjoint (dSOA) method to be competitive computationally with finite differences, as confirmed by the computational results in [12, 22]. However, we will show that the method proposed in this paper is capable of computing accurate directional second order derivatives noticeably cheaper than finite differences. Furthermore, the choice of  $\varepsilon$  in (1.10) is constrained by the need for accurate derivative approximations, on the one hand, and the integration tolerance of the first order adjoint code, on the other hand. Often it is difficult or impossible to find a suitable choice of  $\varepsilon$  for a particular problem. In contrast, the dSOA method described in this paper computes derivatives to the accuracy of the integration tolerance, a numerical parameter that is easy to set and adjust.

In this paper we extend the earlier second order adjoint analysis results to general second order directional derivatives of stiff ODE embedded functionals. For stiff ODEs, differentiation of the entire stiff integration scheme is unreasonable considering the complex code structure of established methods for stiff integration. We propose a numerical procedure that combines AD with numerical integration methods that exploit the structure of the forward and adjoint sensitivity systems by leaving the code for integration intact and constructing sensitivity and adjoint systems individually. This targeted AD approach employing the “differentiate then discretize” strategy allows us to avoid differentiation of the corrector iteration within a multistep integration scheme and to take advantage of the staggered corrector method for large systems. Our strategy applies equally well to nonstiff ODEs. However, the absence of a corrector iteration means that the computational efficiency will remain comparable with a “discretize than differentiate” strategy using AD. On the other hand, our approach may remain attractive when compared with AD because it obviates the need to differentiate any error control mechanism present and the implied difficulties in controlling error in the computed derivatives.

In section 2, we provide a general derivation of the second order adjoint system to obtain directional second order information and give alternative derivations analogous to AD. Section 3 summarizes the numerical procedure for computing directional second order adjoints (dSOA) and its implementation. Numerical experiments to elucidate the computational advantages of our method are given in section 4, which is followed by our conclusions.

**2. Derivation of the directional second order adjoint system.** The dSOA system can be obtained by differentiating  $\frac{\partial G}{\partial p}$  in (1.6) for a second time with respect to  $p$ :

$$\begin{aligned}
 \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} \left\{ g_{pp} + g_{px}x_p - \left[ F_p^T \lambda_p + (\lambda^T \otimes I_{n_p}) \frac{\partial F_p}{\partial p} \right] \right\} dt \\
 &\quad - \int_{t_0}^{t_f} [-(g_x \otimes I_{n_p}) + (\lambda^T F_x \otimes I_{n_p}) - (\dot{\lambda}^T \otimes I_{n_p})] x_{pp} dt \\
 &\quad - \int_{t_0}^{t_f} x_p^T \left[ -g_{xp} - g_{xx}x_p + F_x^T \lambda_p + (\lambda^T \otimes I_{n_x}) \frac{\partial F_x}{\partial p} - \dot{\lambda}_p \right] dt \\
 (2.1) \quad &\quad - [(\lambda^T \otimes I_{n_p}) x_{pp} + x_p^T \lambda_p] \Big|_{t_0}^{t_f}.
 \end{aligned}$$

Now we can set the second order adjoint system as

$$\begin{aligned}
 \dot{\lambda}_p - F_x^T \lambda_p &= (\lambda^T \otimes I_{n_x}) \frac{\partial F_x}{\partial p} - g_{xp} - g_{xx}x_p, \\
 (2.2) \quad \lambda_p(t_f) &= 0,
 \end{aligned}$$

which with (1.7) simplifies (2.1) to

$$\begin{aligned}
 \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} \left\{ g_{pp} + g_{px}x_p - [F_p^T \lambda_p + (\lambda^T \otimes I_{n_p})(F_{pp} + F_{px}x_p)] \right\} dt \\
 (2.3) \quad &\quad + [(\lambda^T \otimes I_{n_p}) x_{pp} + x_p^T \lambda_p] \Big|_{t=t_0}.
 \end{aligned}$$

Instead of solving an  $n_x \times n_p$  system of ODEs in (2.2), we can postmultiply it by a direction vector  $u \in \mathbb{R}^{n_p}$  to obtain

$$\begin{aligned}
 \dot{\lambda}_p u - F_x^T \lambda_p u &= (\lambda^T \otimes I_{n_x}) \frac{\partial F_x}{\partial p} u - (g_{xp} + g_{xx}x_p) u, \\
 (2.4) \quad \lambda_p u \Big|_{t=t_f} &= 0,
 \end{aligned}$$

where the  $n_x^2 \times n_p$  matrix  $\frac{\partial F_x}{\partial p}$  has the following entries:

$$(2.5) \quad \frac{\partial F_x}{\partial p} = \begin{bmatrix} \frac{\partial^2 f_1}{\partial p_1 \partial x_1} + \frac{\partial^2 f_1}{\partial x_1^2} \frac{\partial x_1}{\partial p_1} & \cdots & \frac{\partial^2 f_1}{\partial p_{n_p} \partial x_1} + \frac{\partial^2 f_1}{\partial x_1^2} \frac{\partial x_1}{\partial p_{n_p}} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_1}{\partial p_1 \partial x_{n_x}} + \frac{\partial^2 f_1}{\partial x_{n_x}^2} \frac{\partial x_{n_x}}{\partial p_1} & \cdots & \frac{\partial^2 f_1}{\partial p_{n_p} \partial x_{n_x}} + \frac{\partial^2 f_1}{\partial x_{n_x}^2} \frac{\partial x_{n_x}}{\partial p_{n_p}} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_{n_x}}{\partial p_1 \partial x_1} + \frac{\partial^2 f_{n_x}}{\partial x_1^2} \frac{\partial x_1}{\partial p_1} & \cdots & \frac{\partial^2 f_{n_x}}{\partial p_{n_p} \partial x_1} + \frac{\partial^2 f_{n_x}}{\partial x_1^2} \frac{\partial x_1}{\partial p_{n_p}} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_{n_x}}{\partial p_1 \partial x_{n_x}} + \frac{\partial^2 f_{n_x}}{\partial x_{n_x}^2} \frac{\partial x_{n_x}}{\partial p_1} & \cdots & \frac{\partial^2 f_{n_x}}{\partial p_{n_p} \partial x_{n_x}} + \frac{\partial^2 f_{n_x}}{\partial x_{n_x}^2} \frac{\partial x_{n_x}}{\partial p_{n_p}} \end{bmatrix}.$$

Now by defining  $\mu \equiv \lambda_p u$ , the dSOA equations become

$$(2.6) \quad \begin{aligned} \dot{\mu} - F_x^T \mu &= (\lambda^T \otimes I_{n_x})(F_{x_p} u + F_{x_x}(x_p u)) - g_{xx}(x_p u) - g_{xp} u, \\ \mu(t_f) &= 0. \end{aligned}$$

Hence we can calculate the directional second order derivative of  $G(p)$  by

$$(2.7) \quad \begin{aligned} \frac{\partial^2 G}{\partial p^2} u &= \int_{t_0}^{t_f} \left\{ g_{pp} u + g_{px}(x_p u) - [F_p^T \mu + (\lambda^T \otimes I_{n_p})(F_{pp} u + F_{px}(x_p u))] \right\} dt \\ &+ [(\lambda^T \otimes I_{n_p})x_{pp} u + x_p^T \mu]_{t=t_0}, \end{aligned}$$

where  $x_p u$  is obtained by solving the directional version of the first order forward sensitivity equations (1.2) with  $s \equiv x_p u$ ,

$$(2.8) \quad \dot{s} + \frac{\partial F}{\partial x} s + \frac{\partial F}{\partial p} u = 0, \quad s(t_0) = \frac{\partial x_0}{\partial p} u.$$

For a system of ODEs defined as  $\dot{x} + F(t, x) = 0, x(t_0) = x_0(p)$ , the directional second order derivative reduces to

$$(2.9) \quad \frac{\partial^2 G}{\partial p^2} u = \int_{t_0}^{t_f} [g_{pp} u + g_{px} s] dt + [(\lambda^T \otimes I_{n_p})x_{pp} u + x_p^T \mu] |_{t=t_0}.$$

A more specific case can be constructed when  $\frac{\partial g_p}{\partial p} \equiv g_{pp} + g_{px} x_p = 0$  and

$$(2.10) \quad x_p^T |_{t=t_0} = I_{n_x}.$$

Now the exact Hessian-vector multiplication can be evaluated by calculating the second order adjoint at the initial time. This specific case is presented in [21, 22].

Similar to the first order adjoint case, the directional second order derivative of a point-form functional can be calculated using the identity  $\frac{\partial^2 g}{\partial p^2} |_{t_f, p} \equiv \frac{d}{dt_f} \frac{\partial^2 G}{\partial p^2} |_p$ . Then the first order adjoint equation becomes

$$(2.11) \quad \begin{aligned} \dot{\lambda}_{t_f}^T - \lambda_{t_f}^T F_x &= 0, \\ \lambda_{t_f}(t_f) &= g_x(t_f), \end{aligned}$$

where  $\lambda_{t_f} \equiv \frac{d}{dt_f} \lambda$ . By defining  $\mu_{t_f} \equiv (\lambda_p)_{t_f} u$ , the dSOA equations become

$$(2.12) \quad \begin{aligned} \dot{\mu}_{t_f} - F_x^T \mu_{t_f} &= (\lambda_{t_f}^T \otimes I_{n_x})(F_{x_p} u + F_{x_x} s), \\ \mu(t_f)_{t_f} &= g_{xx}(t_f) s(t_f) - g_{xp}(t_f) u. \end{aligned}$$

Finally, the directional second order derivative of a point-form functional is obtained from

$$\begin{aligned}
 \frac{\partial^2 g}{\partial p^2} u &= \int_{t_0}^{t_f} -\{F_p^T \mu_{t_f} + (\lambda_{t_f}^T \otimes I_{n_p})(F_{pp} u + F_{px} s)\} dt \\
 &\quad + g_{pp}(t_f) u + g_{px}(t_f) s(t_f) \\
 (2.13) \quad &\quad + \left[ (\lambda_{t_f}^T \otimes I_{n_p}) x_{pp} u + x_p^T \mu_{t_f} \right]_{t=t_0}.
 \end{aligned}$$

The formulation of the first order (1.7) and directional second order (2.6) equations, directional first order sensitivity equations (2.8), along with the quadrature equation (2.7) requires several vector-matrix, matrix-vector, and vector-matrix-vector products, i.e.,

$$\begin{aligned}
 &F_x s + F_p u, \\
 &\quad \lambda^T F_x, \\
 &\quad F_x^T \mu, \\
 &(\lambda^T \otimes I_{n_x})(F_{xp} u + F_{xx} s), \\
 &\quad g_{xx} s - g_{xp} u, \\
 &\quad g_{pp} u + g_{px} s, \\
 &F_p^T \mu + (\lambda^T \otimes I_{n_p})(F_{pp} u + F_{px} s), \\
 &\quad (\lambda^T \otimes I_{n_p}) x_{pp} u + x_p^T \mu.
 \end{aligned}$$

These terms should be calculated accurately and efficiently for a successful implementation of the dSOA method.

**2.1. Obtaining first and second order derivative information via AD.**

First and second order derivative information of vector functions which are a composition of twice continuously differentiable elementary functions can be obtained using the forward and reverse modes of AD [9]. The forward mode of calculating the first derivative vectors gives the tangents  $\dot{y} = F'(x)\dot{x}$  for a certain seed direction  $\dot{x}$ , whereas the reverse mode gives the gradients  $\bar{x} = \bar{y}F'(x)$  for the weight functions  $\bar{y}$ . The cost of the forward propagation for the tangents will increase linearly with  $n_p$ , the number of parameters (also, the number of domain directions  $\dot{x}$  along which we want to differentiate). The cost for the reverse (adjoint) mode, on the other hand, depends on the number of gradients to be evaluated. This result can also be seen from first order adjoint system derivations [3]. To obtain the second order derivative information in the form of Hessian-vector products there are four different options, namely,

1. applying the adjoint mode for the second time to the adjoint mode results (adjoints of adjoints),
2. using the forward mode over the forward mode results (forward over forward mode (FOF) [8]),
3. applying the forward mode to adjoint (reverse) mode results (tangents of adjoints [9] or forward over reverse mode (FOR) [8]),
4. applying the reverse mode to tangent mode results.

An analogous analysis of the methods for the derivation of the dSOA equations is given in Appendix B.

It has been shown [9] that adjoints of adjoints can be realized as tangents of adjoints. Recalling the cheap gradient and cheap Hessian results mentioned in section 1,

the accurate and efficient calculation of the terms required by the method presented in this paper call for an AD tool with forward and reverse mode capabilities to be employed.

**3. Numerical procedure and implementation.** The evaluation of the second order directional derivatives of stiff ODE embedded functionals requires the solution of the state equations and directional first order sensitivities forward in time:

$$\begin{aligned} \dot{x} + F(t, x, p) &= 0, & x(t_0) &= x_0(p), \\ \dot{s} + F_x s + F_p u &= 0, & s(t_0) &= (x_p u)|_{t=t_0}. \end{aligned}$$

At the final time point of the forward integration, initial values for the adjoint variables should be calculated for point-form functionals. Then, first order adjoint and second order directional adjoint equations must be integrated backward in time:

$$\begin{aligned} \dot{\lambda}^T - \lambda^T F_x &= -g_x, & \lambda^T(t_f) &= 0, \\ \dot{\mu} - F_x^T \mu &= (\lambda^T \otimes I_{n_x})(F_{xp} u + F_{xx} s) - g_{xx} s - g_{xp} u, & \mu(t_f) &= 0. \end{aligned}$$

It should be noted that the size of these four systems is independent of  $n_p$ . Finally, depending on the structure of the functional, a quadrature calculation (e.g., via an efficient staggered method [14]) may be necessary to obtain the end result

$$\begin{aligned} \frac{\partial^2 G}{\partial p^2} u &= \int_{t_0}^{t_f} \left\{ g_{pp} u + g_{px} s - [F_p^T \mu + (\lambda^T \otimes I_{n_p})(F_{pp} u + F_{px} s)] \right\} dt \\ &+ [(\lambda^T \otimes I_{n_p})x_{pp} u + x_p^T \mu]_{t=t_0}. \end{aligned}$$

This quadrature involves the calculation of  $n_p$  quantities. This “weak”  $n_p$ -dependence, which is also present in the first order adjoint case (see section 1), should be noted, especially when the cost of solving the state equations (i.e., cost of simulation) is relatively small.

In general, the derivatives  $F_x, F_p, F_{xp}, F_{xx}, F_{px},$  and  $F_{pp}$  in the adjoint systems and quadrature calculations will depend on the state variables  $x$  at each time point of the backward integration. This dependence necessitates the storage and recovery of the forward information to be utilized in the backward integration.

Numerical integration methods for stiff ODEs typically employ a corrector iteration at each time step. Efficient methods exist that can exploit the similarity between the corrector iteration for the state equations and those for the directional first order sensitivities, in particular the staggered corrector method of [5]. This method performs best when the number of directions is much less than  $n_x$ , which is the situation considered here. The second order adjoints can be interpreted as sensitivities of the first order adjoints; therefore they can also be calculated employing a staggered corrector method similar to the forward sensitivities. Again, this will perform best when there is one or a small number of directions.

Considering these issues, the numerical procedure implemented involves a forward integration using the staggered corrector method [5] to obtain the states and directional first order sensitivities in one or a small number of directions. These values are stored. Subsequently, a backward integration is performed of the first order adjoint and second order directional adjoint equations in one or a small number of directions, once again, with the staggered corrector method. Finally,  $\frac{\partial^2 G}{\partial p^2} u$  is evaluated at time  $t_0$  from a quadrature calculation.

We have implemented the above summarized procedure within a modified version of DASPKADJOINT [4, 14], an adjoint sensitivity solver. Specifically, the aforementioned code is modified to accommodate sensitivity calculations during backward integration, to save and retrieve forward sensitivities, to calculate the second order derivatives at the final step, and to perform these calculations for several directions. These modifications are carried out only for the integration methods using direct linear solvers.

The input information necessary to set up a given problem is similar to that of DASPKADJOINT. However, in addition to the state and first order adjoint equations, directional first order sensitivity and dSOA equations along with the integrands of the quadrature are needed.

**4. Numerical examples.** We present three examples to showcase the effectiveness and efficiency of the dSOA method, especially compared to the directional finite difference approach with a first order adjoint code. Our first example is from the optimal control literature. It consists of a small ODE system and a point-form functional. Using a control parameterization discretization of the control, the number of parameters can be increased. The second example is the two-dimensional (2-D) heat equation. The PDE is reduced to a stiff ODE via the numerical method of lines (NMOL). The number of state equations can become large by increasing the mesh. The directional derivative of an integral-form functional is calculated. Boundary values and initial conditions are chosen as parameters along with two additional ones from the PDE. Our final example is an industrial sterilization problem, which contains PDEs from Fourier's second law that are reduced to stiff ODEs with NMOL, coupled with ODEs describing the kinetics of the system. Directional second order derivatives of an integral-form functional are calculated considering boundary values and initial conditions as parameters. Additional parameters are included by a control parameterization.

The performance comparison of dSOA relative to the finite difference method (FDM) and a simulation are presented in various cases with different problem structures, parameter types, and number of parameters, considering the elapsed CPU time as a simple cost measure. It is reasonable to anticipate that the dSOA method will take less than 4 times the time of a simulation if the quadrature calculations do not dominate. This is because it requires one forward simulation combined with a directional sensitivity calculation, which can cost up to twice the cost of a single simulation. Employing a similar reasoning for the backward integration, the cost for dSOA can be anticipated to be between 2 and 4 times the cost of a simulation.

Numerical experiments are performed on a Pentium III/733 MHz IBM machine with 256 MB memory and running Linux kernel 2.4. All AD tasks are performed by the AD tool TAMC [7]. DASPKADJOINT has a checkpointing scheme to reduce the load on the active memory. However, no checkpointing is used for the following examples.

**4.1. van der Pol oscillator.** This example is taken from the optimal control literature [2]. The point-form functional is

$$(4.1) \quad g(x(t_f, p), p) = x_3(t_f, p).$$

The embedded ODE system consists of the three equations

$$\dot{x}_1 - (1 - x_2^2)x_1 + x_2 - v(p) = 0,$$

$$\begin{aligned} \dot{x}_2 - x_1 &= 0, \\ \dot{x}_3 - x_1^2 - x_2^2 - v(p)^2 &= 0, \end{aligned}$$

where the control variable can be parameterized using a piecewise linear function on a uniform time grid ( $\Delta t = t_{i+1} - t_i$ ) of a given time range  $t_1, \dots, t_{n_p}$ :

$$v(t, p) = \sum_{i=1}^{n_p-1} 1_{\mathcal{P}_i}(t) \left[ \left( i + \frac{t_1 - t}{\Delta t} \right) p_i + \left( 1 - i + \frac{t - t_1}{\Delta t} \right) p_{i-1} \right].$$

The indicator function  $1_{\mathcal{P}_i}(t)$  for the  $i$ th partition of the time domain,  $\mathcal{P}_i$ , is given by

$$1_{\mathcal{P}_i}(t) = \begin{cases} 1, & t \in \mathcal{P}_i, \\ 0, & t \notin \mathcal{P}_i. \end{cases}$$

The initial condition for the states is  $\mathbf{x}(0) = (0 \ 1 \ 0)$ , with  $0 \leq t \leq t_f$ , where  $t_f = 5.0$ . For this simple example the directional forward sensitivity equations are

$$\dot{s} + \begin{bmatrix} -(1 - x_2^2) & (2x_2x_1 + 1) & 0 \\ -1 & 0 & 0 \\ -2x_1 & -2x_2 & 0 \end{bmatrix} s + \begin{bmatrix} -v_p \\ 0_{(1 \times p)} \\ -2vv_p \end{bmatrix} u = 0,$$

where  $0_{(1 \times p)}$  denotes a  $1 \times p$  zero matrix. First order adjoint and dSOA equations can be constructed as follows:

$$\dot{\lambda} - \begin{bmatrix} -(1 - x_2^2) & (2x_2x_1 + 1) & 0 \\ -1 & 0 & 0 \\ -2x_1 & -2x_2 & 0 \end{bmatrix}^T \lambda = - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\dot{\mu} - \begin{bmatrix} -(1 - x_2^2) & (2x_2x_1 + 1) & 0 \\ -1 & 0 & 0 \\ -2x_1 & -2x_2 & 0 \end{bmatrix}^T \mu = (\lambda^T \otimes I_3) F_{xx} s,$$

where

$$F_{xx}^T = \begin{bmatrix} 0 & 2x_2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 2x_2 & 2x_1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The quadrature for this example becomes

$$\frac{\partial^2 g}{\partial p^2} u = \int_0^{t_f} \left\{ \begin{bmatrix} -v_p^T & 0_{(p \times 1)} & -2vv_p^T \end{bmatrix} \mu + (\lambda^T \otimes I_{n_p}) \begin{bmatrix} -v_{pp} \\ 0_{(p \times p)} \\ -2(v_p^T v_p + vv_{pp}) \end{bmatrix} u \right\} dt.$$

There are two main criteria with which to assess the performance of our dSOA method. First, the cost of the method can be compared with the cost of a simulation (SIM). Provided that there is enough memory for calculations, one would expect a constant cost ratio  $\frac{\text{cost}(dSOA)}{\text{cost}(SIM)}$  of between 2 and 4 as the number of parameters is increased. Secondly, we can compare the cost of calculating a directional second order derivative with the dSOA method and the FDM. Our implementation of the FDM consists of applying the first order adjoint method consecutively at a nominal

TABLE 4.1  
*Cost comparisons for the van der Pol oscillator example.*

$n_p$	$cost(SIM)$ (CPU (s))	$\frac{cost(dSOA)}{cost(SIM)}$	$\frac{cost(dSOA)}{cost(FDM)}$
1	0.01	1.0	0.67
10	0.01	2.0	0.68
100	0.04	5.4	0.53
1000	0.22	21.1	0.48
2000	0.40	45.1	0.53

point and at a perturbed point in the given direction. A cost ratio  $\frac{cost(dSOA)}{cost(FDM)}$  less than 1.0 would mean that the dSOA method calculated the directional second order derivatives more cheaply than the FDM. In other words, we can obtain second order information not only with controlled accuracy but also cheaply.

Because the control vector parameterization for this example is done using a piecewise linear approximation on a uniform partition of the time interval, the possibility of any discontinuity in the first and second order adjoint system should be considered. A short argument demonstrating that one would not expect any “jumps” in this example is given in Appendix C.

For test runs the following direction vector is chosen:

$$u = \left( \frac{1}{n_p}, \dots, \frac{1}{n_p} \right),$$

which corresponds to a linear combination of all columns in the  $\frac{\partial^2 g}{\partial p^2} \big|_{p=p^*}$  matrix, where  $p^*$  is

$$p^* = (0.7, \dots, 0.7).$$

For this example, the relative (RTOL) and absolute (ATOL) tolerances for the integration are both  $10^{-7}$ .

In Table 4.1 a cost comparison between a simulation, dSOA, and FDM methods presents two important results relevant to the dSOA method and other approaches to calculating derivative information. The encouraging conclusion is that by the dSOA method the second order directional derivative can be calculated much more cheaply than via the FDM. However, the cost of dSOA and obviously of FDM (considering  $\frac{cost(FDM)}{cost(SIM)} = \frac{cost(dSOA)}{cost(SIM)} \left( \frac{cost(dSOA)}{cost(FDM)} \right)^{-1}$ ) does not stay constant within the range 2–4. This odd observation is in fact reasonable because of the “weak” dependence of the methods on the number of parameters. Whenever the number of the parameters is much larger than the number of state variables, the cost of calculating the quadratures will dominate. For some problems, by exploiting the problem structure the cost of quadrature calculations can be drastically reduced. For instance, because of the nature of the piecewise linear control parameterization, only two quadratures have to be calculated at any point in time since all the others will stay constant between two consecutive mesh points. However, in general this reduction in the active quadrature variables is not possible. As can be seen in the case when the control variables are parameterized using Chebyshev polynomials over the whole time domain instead of piecewise linear approximation (Table 4.2), the cost ratio  $\frac{cost(dSOA)}{cost(SIM)}$  will continue to increase proportionally with the number of parameters. This expected result confirms the “weak” dependence of the adjoint methods on the number of parameters

TABLE 4.2  
*Cost comparisons for the van der Pol oscillator example (Chebyshev approximation).*

$n_p$	$cost(SIM)$ (CPU (s))	$\frac{cost(dSOA)}{cost(SIM)}$
1	0.01	1.0
10	0.01	1.0
100	0.01	10.2
1000	0.18	27.5
2000	0.62	28.4

and makes these methods effective mainly for large dynamic systems with many parameters, as shown in the following examples.

**4.2. 2-D heat equation.** This example, adapted from [3], consists of a system of PDEs

$$z_t = p_1 z_{xx} + p_2 z_{yy},$$

posed on a 2-D unit square with zero Dirichlet boundary conditions. The nominal values of  $p_1$  and  $p_2$  are equal to 1.0. Spatial derivatives are approximated by centered finite difference approximations on a uniform grid of size  $M$ . The boundary conditions are included in the discretized PDE, reducing the system to an ODE. The initial conditions are posed as

$$z(0, x, y) = 16x(1-x)y(1-y).$$

The functional we consider is of integral form, namely,

$$G(p) = \int_{t_0}^{t_f} \sum_{i=1}^{n_x} z_i dt,$$

where  $t_0 = 0$  and  $t_f = 0.16$ .

In contrast to the previous example, in the 2-D heat equation problem the number of state variables, along with the parameters, which are the boundary conditions, initial conditions, and two parameters from the original PDE, increase. The direction is chosen to be

$$u_1 = (1, 0, \dots, 0).$$

The RTOL for the integration is  $10^{-8}$ , whereas the ATOL is  $10^{-10}$ .

Since the calculation of the quadratures does not dominate the computational cost, a  $\frac{cost(dSOA)}{cost(SIM)}$  ratio in the expected range 2–4 is observed (see column 4 in Table 4.3). As in the previous example, the cost of calculating a directional second order derivative with dSOA is lower than for the FDM, especially for larger problems (column 5 in Table 4.3).

In some applications, such as biconjugate methods, two directional derivatives can be desired at the same time. The performance of the dSOA method in obtaining the derivatives in two directions is also very promising because the incremental cost of computing the second and subsequent directions in the staggered corrector method is typically much less than that for computing the first direction. For the following

TABLE 4.3  
*Cost comparisons for 2-D heat equation example.*

$n_x$ ( $M \times M$ )	$n_p$	$cost(SIM)$ (CPU (s))	$\frac{cost(dSOA)}{cost(SIM)}$	$\frac{cost(dSOA)}{cost(FDM)}$	$\frac{cost(dSOA_2)}{cost(dSOA_1)}$
1600	1762	5.83	3.7	0.79	1.31
6400	6722	70.55	3.0	0.70	1.24
10000	10402	179.53	2.7	0.67	1.30
22500	23102	932.84	3.0	0.62	1.13

two directions,

$$u_1 = (1, 0, \dots, 0),$$

$$u_2 = \left( \frac{1}{n_p}, \dots, \frac{1}{n_p} \right),$$

when we compare the computational cost of the second order derivatives in these two directions at the same time with the cost of calculating the first column of the Hessian matrix, i.e., in the direction  $u_1$  only, we observe that additional directional derivatives can be obtained even “cheaper” (column 6 in Table 4.3).

**4.3. Industrial sterilization of canned foods.** The problem of industrial sterilization [2, 1] is considered as the third example. This involves modeling heating and thermal degradation processes. Heating of the canned foods by conduction is modeled by employing Fourier’s second law on cylindrical coordinates,

$$T_t = \alpha \left( T_{rr} + \frac{1}{r} T_r + T_{zz} \right),$$

with  $r \in [0, R]$ ,  $z \in [0, L]$ , the boundary conditions

$$T(R, z, t) = v(p, t),$$

$$T(r, L, t) = v(p, t),$$

$$T_r(0, z, t) = 0.0,$$

$$T_z(r, 0, t) = 0.0,$$

and the initial conditions

$$T(r, z, 0) = T_0.$$

At  $r = 0$  the singularity is eliminated by applying L’Hospital’s rule to obtain an equivalent expression. NMOL is used to transform the PDE into a set of ODEs on an  $M \times N$  grid. The spatial derivatives are approximated on a uniform grid by different methods, depending on the physical location of the grid points. At the outermost grid points the derivatives are approximated by

$$\frac{\partial f(x_i)}{\partial x} \approx \frac{f(x_{i+1}) + 2f(x_{i+2}) - f(x_{i+3}) - 2f(x_i)}{2\Delta x}$$

and

$$\frac{\partial^2 f(x_i)}{\partial x^2} \approx -\frac{f(x_{i+3}) - 4f(x_{i+2}) - f(x_{i+1}) + 4f(x_i)}{4\Delta x^2}$$

TABLE 4.4  
 Data for the canned food sterilization example.

$C_{N0}$	$1 \times 10^8$
$C_{M0}$	1.0
$T_0$	71.11 °C
$T_{M,ref}$	121.11 °C
$Z_{M,ref}$	10.0 °C
$D_{M,ref}$	240.0 s
$T_{N,ref}$	121.11 °C
$Z_{N,ref}$	25.56 °C
$D_{N,ref}$	10716.0 s
$R$	0.04375 m
$L$	0.058 m
$V^T$	$3.488 \times 10^{-4} \text{ m}^3$
$\alpha$	$1.5443 \times 10^{-7} \text{ m}^2 \text{ s}^{-1}$

for the first order and second order cases, respectively. The second order derivatives at the next interior grid points are approximated by the formula

$$\frac{\partial^2 f(x_i)}{\partial x^2} \approx \frac{f(x_{i+2}) + f(x_{i+1}) - 5f(x_i) + 3f(x_{i-1}))}{4\Delta x^2},$$

whereas the rest of the grid is approximated by centered finite differences. In the above,  $x$  stands for  $r$  or  $z$ .

Pseudo-first order kinetics for the thermal degradation of both the nutrients and microbial spores are described as follows:

$$C_{mt} = - \left( \frac{\ln 10}{D_{m,ref}} \right) C_m \exp \left( \frac{T(r, z, t) - T_{m,ref}}{Z_{m,ref}} \ln 10 \right),$$

$$C_{nt} = - \left( \frac{\ln 10}{D_{n,ref}} \right) C_n \exp \left( \frac{T(r, z, t) - T_{n,ref}}{Z_{n,ref}} \ln 10 \right),$$

with initial conditions  $C_m(0) = C_{m0}$  and  $C_n(0) = C_{n0}$ , respectively.

We consider the final retention of a nutrient as our functional, i.e.,

$$G(p) = \frac{1}{V_T} \int_0^{V_T} \exp \left[ \frac{-\ln 10}{D_{n,ref}} \int_{t_0}^{t_f} \exp \left( \frac{T(r, z, t) - T_{n,ref}}{Z_{n,ref}} \ln 10 \right) dt \right] dV,$$

where  $v(t, p) = T_{retort}(t)$ , describing the parameterized control variable  $T_{retort}$ , with  $t_0 = 0$  and  $t_f = 8000$ . All other parameters defining the canned food sterilization example are given in Table 4.4.

In this case, the number of parameters can be increased in two ways: either by increasing the number of state equations or by applying a finer control parameterization. In this sense, this problem is a combination of the previous two examples and exercises many features of our general dSOA formulation. We obtain the dSOA result at nearly 2.5 times the cost of a single state system simulation (see column 4 in Table 4.5). Similar to the van der Pol oscillator example, the quadrature calculations here become costly as the number of parameters gets larger with respect to the number of state variables, increasing the  $\frac{\text{cost}(dSOA)}{\text{cost}(SIM)}$  ratio. Nevertheless, in general 50–55% of the FDM cost is sufficient for calculating a directional second order derivative (column 5 in Table 4.5). For this example, we have used a piecewise linear control

TABLE 4.5  
*Cost comparisons for canned food sterilization example.*

$n_x$ ( $3 \times (M \times N)$ )	$n_p$	$cost(SIM)$ (CPU (s))	$\frac{cost(dSOA)}{cost(SIM)}$	$\frac{cost(dSOA)}{cost(FDM)}$
75	175	3.85	2.2	0.51
75	575	17.60	3.5	0.48
300	400	73.92	2.2	0.53
300	800	333.60	2.6	0.54
1200	1300	1273.40	2.4	0.50

parameterization on a uniform grid (see section 4.1) with constant reference values of 100.0 (i.e.,  $T_{retort} = 100.0$ ). The second order derivative is calculated for the direction

$$u_1 = (1, 0, \dots, 0),$$

at an RTOL of  $10^{-7}$  and ATOL of  $10^{-12}$ .

**5. Conclusions.** A second order adjoint method is proposed for calculating directional second order derivative information for stiff ODE embedded functionals. By applying a targeted version of algorithmic differentiation to construct the governing ODEs instead of its direct application to a discretized code, we can exploit the structure of the state, sensitivity, and adjoint systems within a state-of-the-art stiff ODE and sensitivity solver. This “differentiate then discretize” strategy is shown to achieve a much better performance than the “discretize then differentiate” strategy if the quadrature calculation does not dominate.

The dSOA method is implemented by modification of an existing first order adjoint sensitivity solver, DASPKADJOINT, and used to solve several problems with different structure and size. In all cases, dSOA outperforms the directional FDM based on a first order adjoint code; i.e., accurate error controlled directional second order information can be obtained with dSOA at a much lower computational cost. Moreover, an additional direction can be calculated for an even lower incremental cost. The results presented should prove significant in applications where directional second order information (Hessian-vector products) is required, such as conjugate gradient or Arnoldi methods. Biconjugate methods can benefit from dSOA even more.

A natural extension of the dSOA method to DAE systems is in progress. The solution of dynamic optimization problems via a “dSOA powered” truncated Newton method and the method’s application to hybrid systems [6] are our two main future research directions.

The staggered corrector method offers computational advantages only when a direct linear solver is employed for the corrector iteration. Often, iterative linear solvers are used for the corrector iteration, particularly NMOL discretizations of PDEs. It would be worthwhile to explore dSOA in conjunction with numerical integration schemes more suitable for iterative linear solvers [19].

**Appendix A. Notation for derivatives of vectors and matrices.** Consider an  $(n_x \times 1)$  vector  $F$ ,

$$F = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n_x-1} \\ f_{n_x} \end{bmatrix}.$$

The derivative of vector  $F$  with respect to  $p$  is defined as

$$F_p \equiv \frac{\partial F}{\partial p} = \begin{bmatrix} \frac{\partial f_1}{\partial p_1} & \cdots & \frac{\partial f_1}{\partial p_{n_p}} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_{n_x}}{\partial p_1} & \cdots & \frac{\partial f_{n_x}}{\partial p_{n_p}} \end{bmatrix}.$$

Moreover, the derivative of  $F_p$  with respect to  $x$  is written as

$$F_{px} \equiv \frac{\partial^2 F}{\partial x \partial p} = \begin{bmatrix} \frac{\partial^2 f_1}{\partial x_1 \partial p_1} & \cdots & \frac{\partial^2 f_1}{\partial x_{n_x} \partial p_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_1}{\partial x_1 \partial p_{n_p}} & \cdots & \frac{\partial^2 f_1}{\partial x_{n_x} \partial p_{n_p}} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_{n_x}}{\partial x_1 \partial p_1} & \cdots & \frac{\partial^2 f_{n_x}}{\partial x_{n_x} \partial p_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f_{n_x}}{\partial x_1 \partial p_{n_p}} & \cdots & \frac{\partial^2 f_{n_x}}{\partial x_{n_x} \partial p_{n_p}} \end{bmatrix},$$

which is an  $n_p \cdot n_x \times n_x$  matrix. In addition,

$$\frac{\partial F_x}{\partial p} \equiv F_{xx}x_p + F_{xp}.$$

The Kronecker product is denoted by  $\otimes$ . If  $A$  is an  $r \times s$  and  $B$  an  $t \times u$  matrix, then their Kronecker product, an  $r \cdot t \times s \cdot u$  matrix, is formed by replacing each  $a_{ij}$  in  $A$  by  $a_{ij}B$  [17]. The derivative of matrix products is given as

$$\frac{\partial}{\partial x} A C = (A \otimes I_t) \frac{\partial C}{\partial x} + (I_r \otimes C^T) \frac{\partial A}{\partial x},$$

where  $A$  is an  $r \times s$  and  $C$  an  $s \times t$  matrix.  $I_t$  and  $I_r$  are identity matrices of size  $t$  and  $r$ , respectively.

**Appendix B. Analysis of methods for obtaining directional second order information.** There appear to be four different approaches to computing  $\frac{\partial^2 G}{\partial p^2}u$ , analogous to the options for AD described in section 2.1, namely, adjoint over adjoint, adjoint over forward, forward over forward, and forward over adjoint.

Equations (1.7) and (2.6)–(2.8) describe the *forward over adjoint* method, which requires the forward integration of the state and directional first order sensitivities ( $2 \times n_x$  equations) and a backward integration of the first order adjoint model along with the “directional sensitivities of the adjoint” system (another  $2 \times n_x$  equations). The first order adjoint equation eliminates the  $x_{pp}$  term, and the second order adjoint equation gets rid of the  $x_p^T$  term in (2.1).

The directional second order sensitivity of  $G$  (or  $g$ ) with respect to  $p$  can also be obtained by solving a subset of the forward sensitivity equations for the state variables. This is actually the *forward over forward* method, which requires a system of  $2 \times n_x + n_x \times n_p$  equations to be integrated forward in time, provided that  $g_{xp}u + g_{xs}s = 0$ . In this method, the directional second order sensitivity equation becomes

$$(B.1) \quad \dot{s}_p + F_x s_p + (I_{n_x} \otimes s^T) \frac{\partial F_x}{\partial p} + (I_{n_x} \otimes u^T) \frac{\partial F_p}{\partial p} = 0.$$

Note that  $s_p$  is equal to

$$(B.2) \quad s_p = (I_{n_x} \otimes u^T) x_{pp};$$

therefore, postmultiplying  $s_p$  with the same directional vector  $u$  and defining  $z_p \equiv s_p u$ , equation (B.1) becomes

$$(B.3) \quad \dot{z}_p + F_x z_p + (I_{n_x} \otimes s^T) \frac{\partial F_x}{\partial p} u + (I_{n_x} \otimes u^T) \frac{\partial F_p}{\partial p} u = 0.$$

The directional second order derivative of  $g$  can be calculated using

$$\frac{\partial^2 g}{\partial p^2} u \Big|_{\substack{t=t_f \\ p=p^*}} = [(g_x \otimes I_{n_p}) x_{pp} u + x_p^T (g_{xp} u + g_{xx} s) + g_{pp} u + g_{px} s]_{p=p^*}^{t=t_f},$$

which still requires the computation of matrix  $x_p^T$  (when  $g_{xp}u + g_{xx}s \neq 0$ ) and, moreover,  $x_{pp} u$ . The latter is needed because premultiplication of  $z_p$  with the pseudoinverse of  $(I_{n_x} \otimes u^T)$  does not yield the required term.

For the *adjoint over forward* method the equation for calculating  $\frac{\partial^2 G}{\partial p^2}$  can be written as

$$(B.4) \quad \begin{aligned} \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} [(g_x \otimes I_{n_p}) - \tilde{\mu}^T (F_x \otimes I_{n_p}) + \dot{\tilde{\mu}}^T] x_{pp} dt \\ &+ \int_{t_0}^{t_f} [x_p^T (g_{xp} + g_{xx} x_p) + g_{pp} + g_{px} x_p + (I_{n_x} \otimes x_p^T) (F_{xp} + F_{xx} x_p)] dt \\ &- \int_{t_0}^{t_f} (\tilde{\mu}^T F_{pp} - \tilde{\mu}^T F_{px} x_p) dt, \end{aligned}$$

which does not eliminate the  $x_p^T$  term and introduces an  $(n_x \cdot n_p \times n_p)$  matrix  $\tilde{\mu}$ .

Finally, the *adjoint over adjoint* method yields

$$(B.5) \quad \begin{aligned} \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} [g_{px} x_p + g_{pp} + x_p^T (g_{xp} + g_{xx} x_p) + (g_x \otimes I_{n_p}) x_{pp}] dt \\ &+ \int_{t_0}^{t_f} (F_p + F_x x_p + \dot{x}_p)^T \mu_x dt \\ &+ \int_{t_0}^{t_f} \mu_\lambda^T [\dot{\lambda}_p - F_x^T \lambda_p - (\lambda^T \otimes I_{n_x}) (F_{xx} x_p + F_{xp}) + g_{xp} + g_{xx} x_p] dt, \end{aligned}$$

which is equivalent to

$$(B.6) \quad \begin{aligned} \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} [g_{px} x_p + g_{pp} + x_p^T (g_{xp} + g_{xx} x_p) + (g_x \otimes I_{n_p}) x_{pp}] dt \\ &+ \int_{t_0}^{t_f} (F_p^T \mu_x + x_p^T F_x^T \mu_x - x_p^T \dot{\mu}_x) dt + (x_p^T \mu_x) |_{t_0}^{t_f} \\ &+ \int_{t_0}^{t_f} [-\dot{\mu}_\lambda^T \lambda_p - \mu_\lambda^T F_x^T \lambda_p - \mu_\lambda^T (\lambda^T \otimes I_{n_x}) (F_{xx} x_p + F_{xp}) \\ &+ \mu_\lambda^T (g_{xp} + g_{xx} x_p)] dt + (\mu_\lambda^T \lambda_p) |_{t_0}^{t_f} \end{aligned}$$

or

$$\begin{aligned}
 \frac{\partial^2 G}{\partial p^2} &= \int_{t_0}^{t_f} [g_{px}x_p + g_{pp} + x_p^T (g_{xp} + g_{xx}x_p) + (g_x \otimes I_{n_p})x_{pp}] dt \\
 &+ \int_{t_0}^{t_f} (-x_p^T \dot{\mu}_x - \mu_\lambda^T F_x^T \lambda_p - \mu_\lambda^T (\lambda^T \otimes I_{n_x})(F_{xx}x_p + F_{xp}) + \mu_\lambda^T (g_{xp} + g_{xx}x_p)) dt \\
 &+ \int_{t_0}^{t_f} (-\dot{\mu}_\lambda^T \lambda_p + F_p^T \mu_x + x_p^T F_x^T \mu_x) dt \\
 \text{(B.7)} \quad &+ (\mu_\lambda^T \lambda_p) \Big|_{t_0}^{t_f} + (x_p^T \mu_x) \Big|_{t_0}^{t_f}.
 \end{aligned}$$

If  $\mu_x = \lambda_p$  and  $\mu_\lambda = -x_p$ , equation (B.7) yields the second order adjoint equations and first order sensitivity equations. After also applying the direct derivation in section 2, we achieve the same result as in the *forward over adjoint* method.

Neither *forward over forward* nor *adjoint over forward* methods can avoid the calculation of the full first order sensitivity matrix ( $x_p^T$ ). On the other hand, *forward over adjoint* and *adjoint over adjoint* methods for the derivation of the directional second order derivatives result in equivalent formulations eliminating the scaling with the number of parameters. From a numerical implementation perspective, the *forward over adjoint* method is favored since it enables efficient forward sensitivity calculation methods to be exploited.

**Appendix C. About the second order information for discontinuous systems.** A scalar valued function is given,

$$\text{(C.1)} \quad G(p) = \int_{t_0}^{t_f} g(t, x(t, p), p) dt,$$

which is related to the ODE system

$$\text{(C.2)} \quad \dot{x} + F(t, x, p) = 0, \quad t_0 \leq t \leq t_f, \quad x(t_0 + 0) = x_0(t_0, p),$$

where

$$\text{(C.3)} \quad F(\cdot) \equiv F_i(t, x, p), \quad t_{i-1} \leq t < t_i, \quad i = 1, 2, \dots, n,$$

$$\text{(C.4)} \quad F(\cdot) = F_{n+1}(t, x, p), \quad t_n \leq t \leq t_f.$$

The “switching conditions” are given by continuously differentiable functions

$$\text{(C.5)} \quad h_i(t_i, x_i(t_i - 0), p) = 0, \quad i = 1, 2, \dots, n, \quad t_0 = t_0(p).$$

Let us define the vector  $x$  at switching instants by the equations

$$\begin{aligned}
 x_i(t_i + 0) &= x_i(t_i - 0) + \Delta_i(t_i, x_i(t_i - 0), p), \quad i = 1, 2, \dots, n, \\
 \text{(C.6)} \quad x_0(t_0 + 0) &= x_0(t_0, p),
 \end{aligned}$$

where  $\Delta_i(\cdot)$  are also continuously differentiable vector functions. The derivation of the first order sensitivities of the functional with respect to the parameters, i.e.,  $\frac{\partial G}{\partial p}$ , provides the first order adjoint equations and the switching conditions of the adjoints [18]. These switching conditions are

$$\begin{aligned}
 \lambda^T(t_i - 0) &= \left\{ \lambda^T(t_i + 0) \left[ I + \frac{\partial \Delta_i}{\partial x_i(t_i - 0)} + \left( \frac{\partial \Delta_i}{\partial t_i} - F_{i+1}(t_i + 0) \right) a_i \right] \right. \\
 \text{(C.7)} \quad &\left. + (g(t_i - 0) - g(t_i + 0)) a_i \right\} (I - F_i(t_i - 0) a_i)^{-1}, \quad i = 1, 2, \dots, n,
 \end{aligned}$$

where  $I$  is the identity matrix and  $a_i$  is

$$(C.8) \quad a_i = - \left( \frac{\partial h_i(\cdot)}{\partial t_i} \right)^{-1} \left( \frac{\partial h_i(\cdot)}{\partial x_i(t_i - 0)} \right).$$

$\frac{\partial G}{\partial p}$  for the discontinuous system differs from that of the continuous one by the following additional summands [18]:

$$(C.9) \quad \sum_{i=1}^n \left\{ \left[ g(t_i - 0) + \lambda^T(t_i - 0)F_i(t_i - 0) - g(t_i + 0) - \lambda^T(t_i + 0)F_{i+1}(t_i + 0) + \lambda^T(t_i + 0) \frac{\partial \Delta_i}{\partial t_i} \right] b_i + \lambda^T(t_i + 0) \frac{\partial \Delta_i}{\partial p} \right\},$$

where

$$(C.10) \quad b_i = - \left( \frac{\partial h_i(\cdot)}{\partial t_i} \right)^{-1} \left( \frac{\partial h_i(\cdot)}{\partial p} \right).$$

Consider a system where the state variables are continuous, i.e.,  $\Delta_i = 0$ , and the switching conditions are

$$(C.11) \quad h_i(t_i, x_i(t_i - 0), p) = t_i - \frac{t_f - t_0}{n_p - 1} = 0, \quad i = 1, 2, \dots, n.$$

Then we will have  $a_i = 0$ ,  $b_i = 0$ , which implies that the adjoints for this system are also continuous (see (C.7)). Since the second order adjoints are equivalent to the sensitivities of the first order adjoints, they will also be continuous, provided that “the event time does not depend on the parameters” [6]. For the second order adjoints to have jumps when the first order adjoints are continuous,  $b_i$  should be nonzero.

**Acknowledgment.** We would like to thank Julio R. Banga for providing the model of the canned food sterilization example.

#### REFERENCES

- [1] J. R. BANGA, R. I. PEREZ-MARTIN, J. M. GALLARDO, AND J. J. CASARES, *Optimization of the thermal processing of conduction-heated canned foods: Study of several objective functions*, J. Food Engineering, 14 (1991), pp. 25–51.
- [2] E. B. CANTO, J. R. BANGA, A. A. ALONSO, AND V. S. VASSILIADIS, *Restricted second order information for the solution of optimal control problems using control vector parameterization*, J. Process Control, 12 (2002), pp. 243–255.
- [3] Y. CAO, S. LI, AND L. PETZOLD, *Adjoint sensitivity analysis for differential-algebraic equations: Algorithms and software*, J. Comput. Appl. Math., 149 (2002), pp. 171–191.
- [4] Y. CAO, S. LI, L. PETZOLD, AND R. SERBAN, *Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution*, SIAM J. Sci. Comput., 24 (2003), pp. 1076–1089.
- [5] W. F. FEEHERY, J. E. TOLSMAN, AND P. I. BARTON, *Efficient sensitivity analysis of large-scale differential-algebraic systems*, Appl. Numer. Math., 25 (1997), pp. 41–54.
- [6] S. GALÁN, W. F. FEEHERY, AND P. I. BARTON, *Parametric sensitivity functions for hybrid discrete/continuous systems*, Appl. Numer. Math., 31 (1999), pp. 17–47.
- [7] R. GIERING, *Tangent linear and adjoint model compiler: Users manual 1.4*, <http://www.autodiff.com/tamc>, 1999.

- [8] R. GIERING AND T. KAMINSKI, *Using TAMC to generate efficient adjoint code: Comparison of automatically generated code for evaluation of first and second order derivatives to hand written code from the Minpack-2 collection*, in Automatic Differentiation for Adjoint Code Generation, C. Faure, ed., INRIA, Sophia Antipolis, France, 1998, pp. 31–37.
- [9] A. GRIEWANK, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in Appl. Math. 19, SIAM, Philadelphia, 2000.
- [10] E. J. HAUG AND P. E. EHLE, *Second-order design sensitivity analysis of mechanical system dynamics*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1699–1717.
- [11] A. C. HINDMARSH AND R. SERBAN, *User Documentation for CVODES: An ODE Solver with Sensitivity Analysis Capabilities*, Technical Report UCRL-MA-148813, Lawrence Livermore National Laboratory, Livermore, CA, 2002.
- [12] F.-X. LE DIMET, I. M. NAVON, AND D. N. DAESCU, *Second-order information for data assimilation*, Monthly Weather Rev., 130 (2002), pp. 629–648.
- [13] S. LI AND L. PETZOLD, *Software and algorithms for sensitivity analysis of large-scale differential algebraic systems*, J. Comput. Appl. Math., 125 (2000), pp. 131–145.
- [14] S. LI AND L. PETZOLD, *Description of DASPKADJOINT: An Adjoint Sensitivity Solver for Differential-Algebraic Equations*, Technical Report, Department of Computer Science, University of California, Santa Barbara, CA, 2001.
- [15] S. LI, L. PETZOLD, AND W. ZHU, *Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem*, Appl. Numer. Math., 32 (2000), pp. 161–174.
- [16] T. MALY AND L. PETZOLD, *Numerical methods and software for sensitivity analysis of differential algebraic equations*, Appl. Numer. Math., 20 (1996), pp. 57–79.
- [17] W. H. MARLOW, *Mathematics for Operations Research*, John Wiley & Sons, New York, 1978.
- [18] A. I. RUBAN, *Sensitivity coefficients for discontinuous dynamic systems*, J. Comput. Systems Sci. Internat., 36 (1997), pp. 536–542.
- [19] M. D. TOCCI, *Sensitivity analysis of large-scale time dependent PDEs*, Appl. Numer. Math., 37 (2001), pp. 109–125.
- [20] V. S. VASSILIADIS, E. B. CANTO, AND J. R. BANGA, *Second-order sensitivities of general dynamic systems with application to optimal control problems*, Chemical Engineering Sci., 54 (1999), pp. 3851–3860.
- [21] Z. WANG, I. M. NAVON, F. X. LE DIMET, AND X. ZOU, *The second order adjoint analysis: Theory and application*, Meteorology and Atmospheric Phys., 50 (1992), pp. 3–20.
- [22] Z. WANG, I. M. NAVON, X. ZOU, AND F. X. LE DIMET, *A truncated Newton optimization algorithm in meteorology applications with analytic Hessian/vector products*, Comput. Optim. Appl., 4 (1995), pp. 241–262.