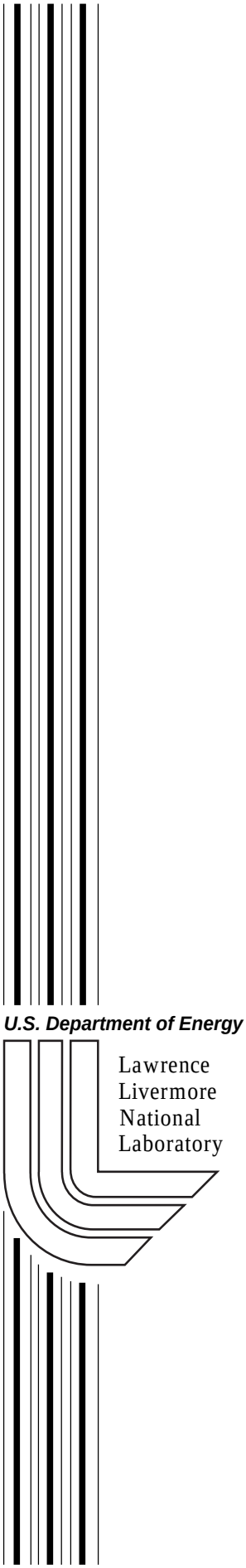
Preprint
UCRL-JP-200039

CVODES: An ODE Solver with Sensitivity Analysis Capabilities

Radu Serban and Alan C. Hindmarsh

This article was submitted to
ACM Transactions on Mathematical Software

September 2003



CVODES: An ODE solver with sensitivity analysis capabilities

RADU SERBAN and ALAN C. HINDMARSH
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

CVODES, which is part of the SUNDIALS software suite, is a stiff and nonstiff ordinary differential equation initial value problem solver with sensitivity analysis capabilities. CVODES is written in a data-independent manner, with a highly modular structure to allow incorporation of different preconditioning and/or linear solver methods. It shares with the other SUNDIALS solvers several common modules, most notably the generic kernel of vector operations and a set of generic linear solvers and preconditioners.

CVODES solves the IVP by one of two methods – backward differentiation formula or Adams-Moulton – both implemented in a variable-step, variable-order form. The forward sensitivity module in CVODES implements the simultaneous corrector method, as well as two flavors of staggered corrector methods. Its adjoint sensitivity module provides a combination of checkpointing and cubic Hermite interpolation for the efficient generation of the forward solution during the adjoint system integration.

We describe the current capabilities of CVODES, its design principles, and connection to the SUNDIALS suite, and the user interface. Finally, we mention current and future development efforts for CVODES, particularly in the direction of automatic generation of the sensitivity right-hand sides using automatic differentiation and/or complex-step techniques.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis; G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—*Initial value problems*; *Multistep methods*; *Stiff equations*

General Terms: Algorithms, Design

Additional Key Words and Phrases: ODEs, Forward Sensitivity Analysis, Adjoint Sensitivity Analysis

1. INTRODUCTION

Fortran solvers for ODE initial value problems (IVPs) are widespread and heavily used. Two solvers that have been written at LLNL in the past are VODE [Brown et al. 1989] and VODPK [Byrne 1992]. VODE is a general purpose solver that includes methods for stiff and nonstiff systems, and in the stiff case uses direct

Authors' address: Radu Serban, Lawrence Livermore National Laboratory, P.O. Box 808, L-560, Livermore, CA 94551; email: radu@llnl.gov; Alan C. Hindmarsh, Lawrence Livermore National Laboratory, P.O. Box 808, L-560, Livermore, CA 94551; email: alanh@llnl.gov;
This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory, under contract No. W-7405-Eng-48.

methods (full or banded) for the solution of the linear systems that arise at each implicit step. Externally, VODE is very similar to the well known solver LSODE [Radhakrishnan and Hindmarsh 1994]. VODPK is a variant of VODE that uses a preconditioned Krylov (iterative) method for the solution of the linear systems. VODPK is a powerful tool for large stiff systems because it combines established methods for stiff integration, nonlinear iteration, and Krylov (linear) iteration with a problem-specific treatment of the dominant source of stiffness, in the form of the user-supplied preconditioner matrix [Brown and Hindmarsh 1989]. The capabilities of both VODE and VODPK have been combined in the C-language packages CVODE [Cohen and Hindmarsh 1996] and PVODE [Byrne and Hindmarsh 1999], later merged under the suite SUNDIALS [Hindmarsh et al. 2003] into one solver, CVODE, which runs on both serial and parallel computers. Besides CVODE, the other two basic solvers in SUNDIALS are IDA, a solver for differential-algebraic equation (DAE) systems, and KINSOL, a Newton-Krylov (GMRES) solver for nonlinear algebraic systems.

In recent years, research and development related to the SUNDIALS solvers has focused on sensitivity analysis to address questions related to unknown parameters in the mathematical models under consideration. Essentially, sensitivity analysis quantifies the relationship between changes in model parameters and changes in model outputs. Such information is crucial for design optimization, parameter estimation, optimal control, data assimilation, process sensitivity, and experimental design. There are two main approaches to sensitivity analysis. *Forward sensitivity analysis* has been proven to be very efficient for problems in which the sensitivities of a (potentially) very large number of quantities with respect to relatively few parameters are needed. However, for problems where the number of uncertain parameters is large, the forward sensitivity method becomes computationally intractable. The *adjoint sensitivity method*, also called reverse method, is advantageous in the complementary situation where the sensitivities of a few quantities with respect to a large number of parameters are needed.

SUNDIALS is currently being expanded to include sensitivity-capable variants of all its basic solvers. The first one, CVODES, released in July 2002, is written with a functionality that is a superset of that of CVODE. Sensitivity analysis capabilities, both forward and adjoint, have been added to the main integrator. Enabling forward sensititivity computations in CVODES will result in the code integrating the so-called *sensitivity equations* simultaneously with the original IVP, yielding both the solution and its sensitivity with respect to parameters in the model. Adjoint sensitivity analysis involves integration of the original IVP forward in time followed by the integration of the so-called *adjoint equations* backwards in time. CVODES provides the infrastructure needed to integrate any final-condition ODE dependent on the solution of the original IVP (not only the adjoint system).

Development of CVODES was concurrent with a redesign of the vector operations module (NVECTOR) across the SUNDIALS suite. The key feature of the new NVECTOR module is that it is written in terms of abstract vector operations with the actual vector kernels attached by a particular implementation (such as serial or parallel). This allows writing the SUNDIALS solvers in a manner independent of the actual NVECTOR vector implementation (which can be user-supplied), as

well as allowing more than one NVECTOR module linked into an executable file. This feature is essential in certain sensitivity analysis computations.

The rest of this paper is organized as follows. In Section 2, the algorithms implemented in CVODES for ODE integration and forward and adjoint sensitivity analysis are presented. The CVODES code organization and relationship to SUNDIALS is discussed in Section 3, while Section 4 gives a high-level overview of the solver usage and general philosophy of the user interface. We conclude with indications on software availability in Section 5 and with some final remarks and directions of current and future development in Section 6.

2. ALGORITHMS

CVODES solves initial value problems for systems of ODEs. Such problems can be stated as

$$\dot{y} = f(t, y)\,, \quad y(t_0) = y_0\,, \tag{1}$$

where $y \in \mathbf{R}^N$ and $\dot{y} = dy/dt$. That is, (1) represents a system of N ordinary differential equations and their initial conditions at some t_0. The dependent variable is y and the independent variable is t. The independent variable need not appear explicitly in the vector valued function f. The vector y will be referred to as the vector of state variables, to distinguish it from sensitivity variables introduced below.

Additionally, if (1) depends (through its right-hand side and/or its initial conditions) on some parameters $p \in \mathbf{R}^{N_p}$, i.e.,

$$\dot{y} = f(t, y, p)\,, \quad y(t_0) = y_0(p)\,, \tag{2}$$

CVODES can also compute first order derivative information, performing either *forward sensitivity analysis* or *adjoint sensitivity analysis*. In the first case, CVODES computes the sensitivities of the solution with respect to the parameters p, while in the second case, CVODES computes the gradient of a *derived function* with respect to the parameters p.

In the rest of this section we describe the algorithms implemented in CVODES, with emphasis on sensitivity analysis. We give only a brief overview of the ODE integration algorithm to introduce some of the quantities needed in the sequel. Since CVODES shares the main integration engine with CVODE, the interested reader is directed to [Hindmarsh et al. 2003].

2.1 ODE Integration

The IVP is solved by one of two numerical methods. These are the backward differentiation formula (BDF) and the Adams-Moulton formula. Both are implemented in a variable-stepsize, variable-order form. The BDF uses a fixed-leading-coefficient form. These formulas can both be represented by a linear multistep formula

$$\sum_{i=0}^{K_1} \alpha_{n,i} y_{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{y}_{n-i} = 0 \tag{3}$$

where the N-vector y_n is the computed approximation to $y(t_n)$, the exact solution of (1) at t_n. The stepsize is $h_n = t_n - t_{n-1}$. The coefficients $\alpha_{n,i}$ and $\beta_{n,i}$ are uniquely

determined by the particular integration formula, the history of the stepsize, and the normalization $\alpha_{n,0} = -1$. The Adams-Moulton formula is recommended for nonstiff ODEs and is represented by (3) with $K_1 = 1$ and $K_2 = q-1$. The order of this formula is q and its values range from 1 through 12. For stiff ODEs, BDF should be selected and is represented by (3) with $K_1 = q$ and $K_2 = 0$. For BDF, the order q may take on values from 1 through 5. In the case of either formula, the integration begins with $q = 1$, and after that q varies automatically and dynamically.

For either BDF or the Adams formula, $\dot{y}_n$ denotes $f(t_n, y_n)$. That is, (3) is an implicit formula, and the nonlinear equation

$$\begin{aligned} G(y_n) &\equiv y_n - h_n\beta_{n,0} f(t_n, y_n) - a_n = 0 \\ a_n &= \sum_{i>0} (\alpha_{n,i} y_{n-i} + h_n \beta_{n,i} \dot{y}_{n-i}) \end{aligned} \tag{4}$$

must be solved for y_n at each time step. For nonstiff problems, a functional (or fixpoint) iteration is normally used which does not require the solution of a linear system of equations. For stiff problems, a Newton iteration is used and for each iteration an underlying linear system must be solved. This linear system of equations has the form

$$M[y_{n(m+1)} - y_{n(m)}] = -G(y_{n(m)}) \,, \tag{5}$$

where $y_{n(m)}$ is the mth approximation to y_n, and M approximates $\partial G/\partial y$:

$$M \approx I - \gamma J, \quad J = \frac{\partial f}{\partial y}, \quad \gamma = h_n \beta_{n,0} \,. \tag{6}$$

At present, aside from a diagonal Jacobian approximation, the other options implemented in CVODES for solving the linear systems (5) are: (a) a direct method with dense treatment of the Jacobian, (b) a direct method with band treatment of the Jacobian, and (c) an iterative method SPGMR (scaled, preconditioned GMRES) [Brown and Hindmarsh 1989], which is a Krylov subspace method. In most cases, performance of SPGMR is improved by user-supplied preconditioners. The user may precondition the system on the left, on the right, on both the left and right, or use no preconditioner. In most cases of interest to the CVODES user, the technique of integration will involve BDF and the Newton method coupled with one of the linear solver modules.

The integrator computes an estimate E_n of the local error at each time step and strives to satisfy the following inequality

$$\|E_n\|_{\mathrm{WRMS}} < 1 \,,$$

where $\|\cdot\|_{\mathrm{WRMS}}$ is the weighted root-mean-square norm defined in terms of the user-defined relative and absolute tolerances. Since these tolerances define the allowed error per step, they should be chosen conservatively. Experience indicates that a conservative choice yields a more economical solution than error tolerances that are too large. The error control mechanism in CVODES varies the stepsize and order in an attempt to take minimum number of steps while satisfying the local error test.

CVODES also incorporates an algorithm for special treatment of quadratures depending on the solution y of the (1) or (2). Evaluation of integrals of the form

$G = \int_{t_0}^{t_f} g(t, y, p)dt$ can be done efficiently using the underlying linear multistep method interpolating polynomials by appending to (2) an additional ODE

$$\dot{\phi} = g(t, y, p)\,, \quad \phi(t_0) = 0\,, \tag{7}$$

in which case $G = \phi(t_f)$. In the context of an implicit ODE integrator, since the right-hand side of (7) does not depend on ϕ, such equations need not participate in the solution of the nonlinear system (4). CVODES allows the user to identify these equations separately from those in (2) and provides the option of including or excluding ϕ from the error control algorithm. The main reason for including this option in CVODES was the need for efficient quadrature computation in the context of adjoint sensitivity analysis (see Section 2.3).

A complete description of the CVODES integration algorithm, including the nonlinear solver convergence, error control mechanism, and heuristics related to stopping criteria and finite-difference parameter selection, is given in [Hindmarsh et al. 2003].

2.2 Forward Sensitivity Analysis

Typically, the governing equations of complex, large-scale models depend on various parameters, through the right-hand side vector and/or through the vector of initial conditions, as in (2). In addition to numerically solving the ODEs, it may be desirable to determine the sensitivity of the results with respect to the model parameters. Such sensitivity information can be used to estimate which parameters are most influential in affecting the behavior of the simulation or to evaluate optimization gradients (in the setting of dynamic optimization, parameter estimation, optimal control, etc.).

The *solution sensitivity* with respect to the model parameter p_i is defined as the vector $s_i(t) = \partial y(t)/\partial p_i$ and satisfies the following *forward sensitivity equations* (or in short *sensitivity equations*):

$$\dot{s_i} = \frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial p_i}\,, \quad s_i(t_0) = \frac{\partial y_0(p)}{\partial p_i}\,, \tag{8}$$

obtained by applying the chain rule of differentiation to the original ODEs (2).

When performing forward sensitivity analysis, CVODES carries out the time integration of the combined system, (2) and (8), by viewing it as an ODE system of size $N(N_s+1)$, where N_s represents a subset of model parameters p_i, with respect to which sensitivities are desired ($N_s \leq N_p$). However, major efficiency improvements can be obtained by taking advantage of the special form of the sensitivity equations as linearizations of the original ODEs. In particular, for stiff systems, for which CVODES employs a Newton iteration, the original ODE system and all sensitivity systems share the same Jacobian matrix, and therefore the same iteration matrix M in (6).

The sensitivity equations are solved with the same linear multistep formula that was selected for the original ODEs and, if Newton iteration was selected, the same linear solver is used in the correction phase for both state and sensitivity variables. In addition, CVODES offers the option of including (*full error control*) or excluding (*partial error control*) the sensitivity variables from the local error test.

2.2.1 *Forward sensitivity methods.* In what follows we briefly describe three methods that have been proposed for the solution of the combined ODE and sensitivity system for the vector $\hat{y} = [y, s_1, \ldots, s_{N_s}]$. Due to its inefficiency, especially for large-scale problems, the first approach is not implemented in CVODES.

Staggered Direct. In this approach [Caracotsios and Stewart 1985], the nonlinear system (4) is first solved and, once an acceptable numerical solution is obtained, the sensitivity variables at the new step are found by directly solving (8) after the BDF discretization is used to eliminate $\dot{s}_i$. Although the system matrix of the above linear system is based on exactly the same information as the matrix M in (6), it must be updated and factored at every step of the integration, in contrast to M which is updated only ocasionally. The computational cost associated with these matrix updates and factorizations makes this method unattractive when compared with the methods described below and is therefore not implemented in CVODES.

Simultaneous Corrector. In this method [Maly and Petzold 1997], the BDF discretization is applied simultaneously to both the original equations (2) and the sensitivity systems (8) resulting in the following nonlinear system

$$\hat{G}(\hat{y}_n) \equiv \hat{y}_n - h_n \beta_{n,0} \hat{f}(t_n,\, \hat{y}_n) - \hat{a}_n = 0\,,$$

where $\hat{f} = [f(t,y,p), \ldots, (\partial f/\partial y)(t,y,p)s_i + (\partial f/\partial p_i)(t,y,p), \ldots]$ and $\hat{a}_n$ are the terms in the BDF discretization that depend on the solution at previous integration steps. This combined nonlinear system can be solved as in (5) using a modified Newton method by solving the corrector equation

$$\hat{M}[\hat{y}_{n(m+1)} - \hat{y}_{n(m)}] = -\hat{G}(\hat{y}_{n(m)}) \tag{9}$$

at each iteration, where

$$\hat{M} = \begin{bmatrix} M & & & & \\ \gamma J_1 & M & & & \\ \gamma J_2 & 0 & M & & \\ \vdots & \vdots & \ddots & \ddots & \\ \gamma J_{N_s} & 0 & \ldots & 0 & M \end{bmatrix},$$

M is defined as in (6), and $J_i = (\partial/\partial y)\,[(\partial f/\partial y)s_i + (\partial f/\partial p_i)]$. It can be shown that 2-step quadratic convergence can be attained by only using the block-diagonal portion of $\hat{M}$ in the corrector equation (9). This results in a decoupling that allows the reuse of M without additional matrix factorizations. However, the products $(\partial f/\partial y)s_i$ as well as the vectors $\partial f/\partial p_i$ must still be reevaluated at each step of the iterative process (9) to update the sensitivity portions of the residual $\hat{G}$.

Staggered corrector. In this approach [Feehery et al. 1997], as in the staggered direct method, the nonlinear system (4) is solved first using the Newton iteration (5).

Then, a separate Newton iteration is used to solve the sensitivity system (8):

$$M[s_{i,n(m+1)} - s_{i,n(m)}] = \\ -\left[s_{i,n(m)} - \gamma\left(\frac{\partial f}{\partial y}(t_n, y_n, p)s_{i,n(m)} + \frac{\partial f}{\partial p_i}(t_n, y_n, p)\right) - a_{i,n}\right], \quad (10)$$

where $a_{i,n} = \sum_{j>0}(\alpha_{n,j}s_{i,n-j} + h_n\beta_{n,j}\dot{s}_{i,n-j})$. In other words, a modified-Newton iteration is used to solve a linear system. In this approach, the vectors $\partial f/\partial p_i$ need be updated only once per integration step, after the state correction phase (5) has converged. Note also that Jacobian-related data can be reused at all iterations (10) to evaluate the products $(\partial f/\partial y)s_i$.

CVODES implements the simultaneous corrector method and two flavors of the staggered corrector method which differ only if the sensitivity variables are included in the error control test. In the *full error control* case, the first variant of the staggered corrector method requires the convergence of the iterations (10) for all N_s sensitivity sytems and then performs the error test on the sensitivity variables. The second variant of the method will perform the error test for each sensitivity vector s_i, $(i = 1, 2, \ldots, N_s)$ individually, as they pass the convergence test. Differences in performance between the two variants may therefore be noticed whenever one of the sensitivity vectors s_i fails a convergence or error test.

An important observation is that the staggered corrector method, combined with the SPGMR linear solver, effectively results in a staggered direct method. Indeed, SPGMR requires only the action of the matrix M on a vector and this can be provided with the current Jacobian information. Therefore, the modified Newton procedure (10) will theoretically converge after one iteration.

2.2.2 *Selection of the absolute tolerances for sensitivity variables.* If the sensitivities are included in the error test, CVODES provides an automated estimation of absolute tolerances for the sensitivity variables based on the absolute tolerance for the corresponding state variable. The relative tolerance for sensitivity variables is set to be the same as for the state variables. The selection of absolute tolerances for the sensitivity variables is based on the observation that the sensitivity vector s_i will have units of $[y]/[p_i]$. With this, the absolute tolerance for the j-th component of the sensitivity vector s_i is set to $\text{ATOL}_j/|\bar{p}_i|$, where ATOL_j are the absolute tolerances for the state variables and $\bar{p}$ is a vector of scaling factors that are dimensionally consistent with the model parameters p and give indication of their order of magnitude. This choice of relative and absolute tolerances is equivalent to requiring that the weighted root-mean-square norm of the sensitivity vector s_i with weights based on s_i is the same as the weighted root-mean-square norm of the vector of scaled sensitivities $\bar{s}_i = |\bar{p}_i|s_i$ with weights based on the state variables (the scaled sensitivities $\bar{s}_i$ being dimensionally consistent with the state variables). However, this choice of tolerances for the s_i may be a poor one, and the user of CVODES can provide different values as an option.

2.2.3 *Evaluation of the sensitivity right-hand side.* There are several methods for evaluating the right-hand side of the sensitivity systems (8): analytic evaluation, automatic differentiation, complex-step approximation, and finite differences (or directional derivatives). CVODES provides all the software hooks for implementing

interfaces to automatic differentiation or complex-step approximation, and future versions will provide these capabilities. At the present time, besides the option for analytical sensitivity right-hand sides (user-provided), CVODES can evaluate these quantities using various finite difference-based approximations to evaluate the terms $(\partial f/\partial y)s_i$ and $(\partial f/\partial p_i)$, or using directional derivatives to evaluate $[(\partial f/\partial y)s_i + (\partial f/\partial p_i)]$. As is typical for finite differences, the proper choice of perturbations is a delicate matter. CVODES takes into account several problem-related features: the relative ODE error tolerance RTOL, the machine unit roundoff U, the scale factor $\bar{p}_i$, and the weighted root-mean-square norm of the sensitivity vector s_i.

Using central finite differences as an example, the two terms $(\partial f/\partial y)s_i$ and $\partial f/\partial p_i$ in the right-hand side of (8) can be evaluated separately:

$$\frac{\partial f}{\partial y}s_i \approx \frac{f(t, y+\sigma_y s_i, p) - f(t, y-\sigma_y s_i, p)}{2\,\sigma_y}\,, \tag{11}$$

$$\frac{\partial f}{\partial p_i} \approx \frac{f(t, y, p+\sigma_i e_i) - f(t, y, p-\sigma_i e_i)}{2\,\sigma_i}\,, \tag{11'}$$

$$\sigma_i = |\bar{p}_i|\sqrt{\max(\text{RTOL}, U)}\,, \quad \sigma_y = \frac{1}{\max(1/\sigma_i, \|s_i\|_{\text{WRMS}}/|\bar{p}_i|)}\,,$$

simultaneously:

$$\frac{\partial f}{\partial y}s_i + \frac{\partial f}{\partial p_i} \approx \frac{f(t, y+\sigma s_i, p+\sigma e_i) - f(t, y-\sigma s_i, p-\sigma e_i)}{2\,\sigma}\,, \tag{12}$$

$$\sigma = \min(\sigma_i, \sigma_y)\,,$$

or adaptively switching between (11)+(11') and (12), depending on the relative size of the estimated finite difference increments σ_i and σ_y.

2.3 Adjoint Sensitivity Analysis

In the *forward sensitivity approach* described in the previous section, obtaining sensitivities with respect to N_s parameters is roughly equivalent to solving an ODE system of size $(1 + N_s)N$. This can become prohibitively expensive, especially for large-scale problems, if sensitivities with respect to many parameters are desired. In this situation, the *adjoint sensitivity method* is a very attractive alternative, provided that we do not need the solution sensitivities s_i, but rather the gradients with respect to model parameters of a relatively few derived functionals of the solution. In other words, if $y(t)$ is the solution of (2), we wish to evaluate the gradient dG/dp of

$$G(p) = \int_{t_0}^{t_\text{f}} g(t, y, p)dt\,, \tag{13}$$

or, alternatively, the gradient dg/dp of the function $g(t, x, p)$ at time t_f. The function g must be smooth enough that $\partial g/\partial y$ and $\partial g/\partial p$ exist and are bounded. In what follows, we only provide the final results for the gradients of both G and $g(t_\text{f})$. For details on the derivation see [Cao et al. 2003]. The gradient of G with respect to p

is simply

$$\frac{dG}{dp} = \lambda^T(t_0)s(t_0) + \int_{t_0}^{t_\mathrm{f}} \left(\frac{\partial g}{\partial p} + \lambda^T \frac{\partial f}{\partial p}\right) dt, \tag{14}$$

where λ is solution of

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial y}\right)^T \lambda - \left(\frac{\partial g}{\partial y}\right)^T, \quad \lambda(t_\mathrm{f}) = 0 \tag{15}$$

and $s(t_0) = dy_0/dp$. The gradient of $g(t_\mathrm{f}, y, p)$ with respect to p can be then obtained by using the Leibnitz differentiation rule. Indeed, from (13), $(dg/dp)(t_\mathrm{f}) = d/dt_\mathrm{f}(dG/dp)$ and therefore, taking into account that dG/dp in (14) depends on t_f both through the upper integration limit and through λ and that $\lambda(t_\mathrm{f}) = 0$,

$$\frac{dg}{dp}(t_\mathrm{f}) = \frac{\partial g}{\partial p}(t_\mathrm{f}) + \mu^T(t_0)s(t_0) + \int_{t_0}^{t_\mathrm{f}} \mu^T \frac{\partial f}{\partial p} dt\,, \tag{16}$$

where μ is the sensitivity of λ with respect to the final integration limit and thus satisfies the following equation, obtained by taking the total derivative with respect to t_f of (15):

$$\dot{\mu} = -\left(\frac{\partial f}{\partial y}\right)^T \mu\,, \quad \mu(t_\mathrm{f}) = \left(\frac{\partial g}{\partial y}(t_\mathrm{f})\right)^T. \tag{17}$$

The final condition on $\mu(t_\mathrm{f})$ follows from $(\partial\lambda/\partial t)+(\partial\lambda/\partial t_\mathrm{f}) = 0$ at t_f, and therefore, $\mu(t_\mathrm{f}) = -\dot{\lambda}(t_\mathrm{f})$.

The first thing to notice about the adjoint system (15) is that there is no explicit specification of the parameters p; this implies that, once the solution λ is found, the formula (14) can then be used to find the gradient of G with respect to any of the parameters p. The same holds true for the system (17) and the formula (16) for gradients of $g(t_\mathrm{f}, y, p)$. The second important remark is that the adjoint systems are terminal value problems which depend on the solution $y(t)$ of the original IVP (2). Therefore, a procedure is needed for providing the states y obtained during a forward integration phase of (2) to CVODES during the backward integration phase of (15) or (17). The approach adopted in CVODES, based on *check-pointing* is described next.

During the backward integration, the evaluation of the right-hand side of the adjoint system requires, at the current time, the states y which were computed in the forward integration phase. Since CVODES implements variable-stepsize integration formulas, it is unlikely that the states will be available at the desired time and therefore some form of interpolation is needed. The CVODES implementation being also variable-order, it is possible that during the forward integration phase the order may be reduced as low as first order, which means that there may be points in time where only y and $\dot{y}$ are available. Therefore, CVODES employs a cubic Hermite interpolation algorithm. However, especially for large-scale problems and long integration intervals, the number and size of the vectors y and $\dot{y}$ that would need to be stored make this approach computationally intractable.

CVODES settles for a compromise between storage space and execution time by implementing a so-called *check-pointing scheme*. At the cost of at most one additional forward integration, this approach offers the best possible estimate of

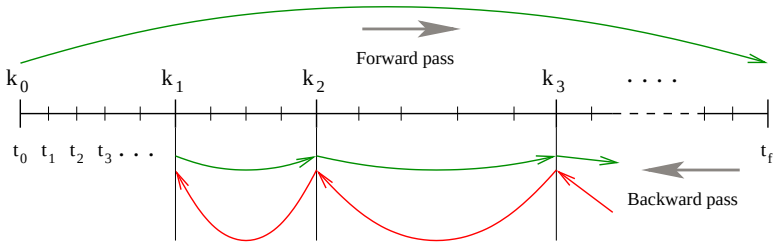


Fig. 1. Illustration of the check-pointing algorithm for generation of the forward solution during the integration of the adjoint system.

memory requirements for adjoint sensitivity analysis. To begin with, based on the problem size N and the available memory, the user decides on the number N_d of data pairs $(y, \dot{y})$ that can be kept in memory for the purpose of interpolation. Then, during the first forward integration stage, every N_d integration steps a check point is formed by saving enough information (either in memory or on disk if needed) to allow for a hot restart, that is, a restart that will exactly reproduce the forward integration. In order to avoid storing Jacobian-related data at each check point, a reevaluation of the iteration matrix is forced before each check point. At the end of this stage, we are left with N_c check points, including one at t_0. During the backward integration stage, the adjoint variables are integrated from t_f to t_0 going from one check point to the previous one. The backward integration from check point $i+1$ to check point i is preceeded by a forward integration from i to $i+1$ during which N_d data pairs $(y, \dot{y})$ are generated and stored in memory for interpolation. This procedure is illustrated in Fig. 1.

This approach transfers the uncertainty in the number of integration steps in the forward integration phase to uncertainty in the final number of check points. However, N_c is much smaller than the number of steps taken during the forward integration, and there is no major penalty for writing and then reading check point data to/from a temporary file. Note that, at the end of the first forward integration stage, data pairs y-$\dot{y}$ are available from the last check point to the end of the integration interval. If no check points are necessary, i.e., N_d is larger than the number of integration steps taken in the solution of (2), the total cost of an adjoint sensitivity computation can be as low as one forward plus one backward integration. In addition, CVODES provides the capability of reusing a set of check points for multiple backward integrations, thus allowing for efficient computation of gradients of several functionals (13).

Finally, we note that the adjoint sensitivity module in CVODES provides the infrastructure to integrate backwards in time any ODE terminal value problem dependent on the solution of the IVP (2), including adjoint systems (15) or (17), as well as any other quadrature ODEs that may be needed in evaluating the integrals in (14) or (16). In particular, for ODE systems arising from semi-discretization of time-dependent PDEs, this feature allows for integration of either the discretized adjoint PDE system or the adjoint of the discretized PDE.

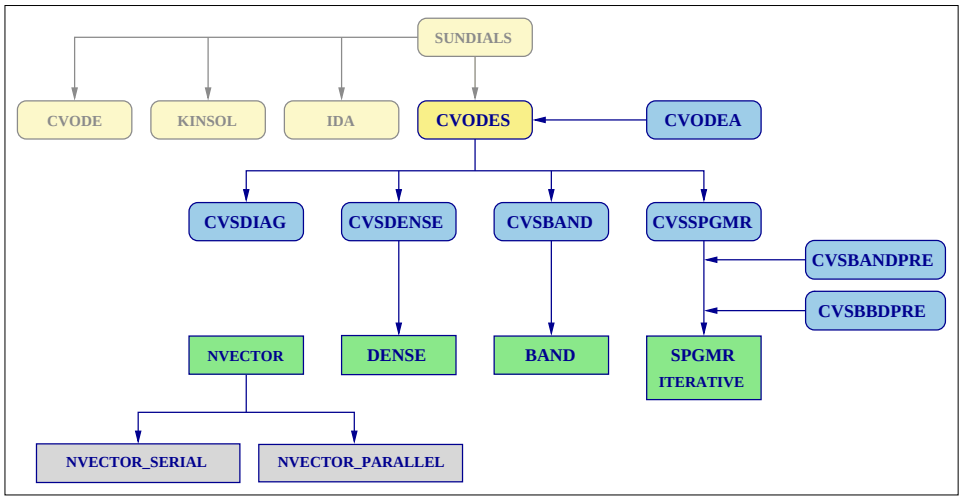


Fig. 2. Overall structure diagram of the CVODES package. Modules specific to CVODES are distinguished by rounded boxes, while generic solver and auxiliary modules are in square boxes.

3. CODE ORGANIZATION

As mentioned before, the SUNDIALS family of solvers consists of CVODE (for ODE systems), KINSOL (for nonlinear algebraic systems), and IDA (for DAE systems). In addition, variants of these which also do sensitivity analysis calculations are available (CVODES) or in development (IDAS and KINSOLS). The overall organization of the CVODES package, as well as its relationship to SUNDIALS, is shown in Fig. 2. The basic elements of the structure are a module for the basic integration algorithm (including forward sensitivity analysis), a module for adjoint sensitivity analysis, and a set of modules for the solution of linear systems that arise in the case of a stiff system.

The central integration module deals with the evaluation of integration coefficients, the functional or Newton iteration process, estimation of local error, selection of stepsize and order, and interpolation to user output points, among other issues. Although this module contains logic for the basic Newton iteration algorithm, it has no knowledge of the method being used to solve the linear systems that arise. For any given user problem, one of the linear system modules is specified and is then invoked as needed during the integration.

In addition, if forward sensitivity analysis is turned on, the main module will integrate the forward sensitivity equations, simultaneously with the original IVP. The sensitivities variables may or may not be included in the local error control mechanism of the main integrator. CVODES provides three different strategies of dealing with the correction stage for the sensitivity variables, simultaneous corrector and two variants of staggered corrector (see Section 2.2). The CVODES package includes an algorithm for the approximation of the sensitivity equations right-hand sides by difference quotients, but the user has the option of supplying these right-hand sides directly.

The adjoint sensitivity module provides the infrastructure needed for the inte-

gration backwards in time of any system of ODEs which depends on the solution of the original IVP, in particular the adjoint system and any quadratures required in evaluating the gradient of the objective functional. This module deals with the set-up of the check points, interpolation of the forward solution during the backward integration, and backward integration of the adjoint equations.

At present, the package includes the following four CVODES linear system modules: (a) CVSDENSE (LU factorization and backsolving with dense matrices), (b) CVSBAND (LU factorization and backsolving with banded matrices), (c) CVSDIAG (an internally generated diagonal approximation to the Jacobian), and (d) CVSSPGMR (scaled preconditioned GMRES method). This set of linear solver modules is intended to be expanded in the future as new algorithms are developed.

In the case of the direct CVSDENSE and CVSBAND methods, the package includes an algorithm for the approximation of the Jacobian by difference quotients, but the user also has the option of supplying the Jacobian (or an approximation to it) directly. In the case of the iterative CVSSPGMR method, the package includes and algorithm for the approximation by difference quotients of the product between the Jacobian matrix and a vector of appropriate length. Again, the user has the option of providing a routine for this operation. In the case of CVSPGMR, the preconditioning must be supplied by the user in two phases: setup (preprocessing of Jacobian data) and solve. While there is no default choice of preconditioner analogous to the difference quotient approximation in the direct case, the references [Brown and Hindmarsh 1989; Byrne 1992], together with the example and demonstration programs included with CVODES, offer considerable assistance in building preconditioners.

Each CVODES linear solver module consists of five routines devoted to: (1) memory allocation and initialization, (2) setup of the matrix data involved, (3) solution of the system, (4) solution of the system in the context of forward sensitivity analysis, and (5) freeing of memory. The setup and solution phases are separate because the evaluation of Jacobians and preconditioners is done only periodically during the integration, as required to achieve convergence. The call list within the central CVODES module to each of the five associated functions is fixed, thus allowing the central module to be completely independent of the linear system method.

These modules are also decomposed in another way. Each of the modules CVSDENSE, CVSBAND, and CVSSPGMR is a set of interface routines built on top of a generic solver module, named DENSE, BAND, and SPGMR, respectively. The interfaces deal with the use of these methods in the CVODES context, whereas the generic solver is independent of the context. While the generic solvers here were generated with SUNDIALS in mind, our intention is that they be usable in other applications as general-purpose solvers. This separation also allows for any generic solver to be replaced by an improved version, with no necessity to revise the CVODES package elsewhere.

CVODES also provides two preconditioner modules. The first one, CVSBANDPRE, is intended to be used on serial computers and provides a banded difference quotient Jacobian based preconditioner and solver routines for use with CVSPGMR. The second preconditioner module, CVBBDPRE, developed for parallel comput-

ers, generates a preconditioner that is a block-diagonal matrix with each block being a band matrix. A detailed description of these two modules, including usage guidelines, is given in [Hindmarsh et al. 2003].

All state information used by CVODES to solve a given problem is saved in a structure, and a pointer to that structure is returned to the user. There is no global data in the CVODES package, and so in this respect it is reentrant. State information specific to the linear solver is saved in a separate structure, a pointer to which resides in the CVODES memory structure. The reentrancy of CVODES was motivated by the anticipated multicomputer extension but is also essential during adjoint sensitivity analysis where the check-pointing algorithm leads to interleaved forward and backward integration passes.

Figure 2 does not show any of the user-supplied routines for CVODES. At a minimum, the user must provide a routine for the evaluation of the ODE right-hand side and, if performing adjoint sensitivity analysis, a routine for the evaluation of the right-hand side of the adjoint system. Optional user-provided routines include, depending on the options chosen, functions for Jacobian evaluation (direct cases) or Jacobian-vector products (Krylov case), setup and solution of Krylov preconditioners, a function providing the integrand of any additional quadrature equations, and a routine for providing the right-hand side of the sensitivity equations (for forward sensitivity analysis). Depending on the options selected for the solution of the adjoint system, the user may have to provide corresponding Jacobian and/or preconditioner routines.

One of the most important characteristics of the design of CVODES (shared by all solvers across SUNDIALS) is the fact that it is implemented in a data-independent manner, in that the solver does not need any information regarding the underlying structure of the data on which it operates.

The CVODES solver acts on vectors through a generic NVECTOR module, which defines an NVECTOR structure specification, a data-independent NVECTOR type, a set of abstract vector operations, and a set of wrappers for accessing the actual vector operations of the implementation under which an NVECTOR was created. Because details of vector operations are thus encapsulated within each specific NVECTOR implementation, CVODES is thus independent of a specific implementation. This allows the solver to be precompiled as a binary library and allows more than one NVECTOR implementation to be used within a single program. This feature is essential for the efficient integration of quadrature variables (see Section 2.1) as well as for adjoint sensitivity analysis when, for some problems, the adjoint variables are more conveniently organized in a structure different from that of the variables in the forward problem.

A particular NVECTOR implementation, such as the serial and parallel implementations included with SUNDIALS or a user-provided implementation, must provide the following: (1) actual implementation of the routines for operations on N-vectors, such as creation, destruction, summation, and dot product; (2) a routine to construct an NVECTOR specification structure for this particular implementation, which defines the data necessary for constructing a new N-vector and attaches the vector operations to the new structure; and (3) a destructor for the NVECTOR specification structure.

4. CVODES USAGE

In this section we give an overview of the usage of CVODES for ODE integration, forward sensitivity analysis, and adjoint sensitivity analysis. Complete documentation of the code usage is given in [Hindmarsh and Serban 2002].

One of the guiding principles in designing the user interface to the CVODES solver has been to allow user to transit from just integration of ODEs to performing sensitivity analysis in as rapid and seamless a manner as possible. To achieve this goal, we have opted not to modify any of the CVODE user interface to account for the initialization and set up of sensitivity analysis.

Instrumenting an existing user code for forward sensitivity analysis can thus be done by only inserting a few calls to CVODES routines, additional to those required for setting up and solving the original ODE (steps 7, 8, 10, and 12 below). We give below the main steps required to set up, initialize, and solve an IVP ODE, and optionally perform forward sensitivity analysis with respect to some of the model parameters. This sequence of calls is the most natural one, but the order of some of the steps below can be changed. For example, initialization and allocation for forward sensitivity analysis could be performed before attaching and configuring the linear solver module. Similarly, changing optional inputs to the solver (step 3) could follow the memory allocation step 4.

(1) An implementation dependent NVECTOR specification constructor must first be called. For the two NVECTOR implementation provided with SUNDIALS, serial and MPI parallel, the constructor routines are `NV_SpecInit_Serial` and `NV_SpecInit_Parallel`, respectively.

(2) `CVodeCreate` creates the solver object. The user must specify the linear multistep method to be used (Adams or BDF) and the nonlinear iteration type (functional or Newton). Various options controlling the solver are set to their default values.

(3) `CVodeSet*` routines can now be used to change various controls from their default values. Choices and default values are given in Table I.

(4) `CVodeMalloc` must be called next to perform any required memory allocation, after checking the initialized memory block for errors in the default or optional inputs. At this step the user must specify the routine providing the ODE right-hand side, the initial time and initial values, as well as the desired integration tolerances.

(5) `CVDense`, `CVBand`, `CVDiag`, or `CVSpgmr` If Newton iteration was selected in step (2), a linear solver is needed for solving the linear systems that arise during the Newton iterations. A direct linear solver (dense, band, diagonal, or SPGMR) must now be created and attached to the block of memory allocated for the solver. Various options controlling the linear solver are set to their default values.

(6) `CVDenseSet*`, `CVBandSet*`, or `CVSpgmrSet*` At this stage, the default values in the linear solver memory block can be changed if so desired. Choices and default values are given in Table I.

(7) `CVodeSetSens*` routines can be called to change from their default values the

optional inputs that control the integration of the sensitivity systems (see Table I).

(8) `CVodeSensMalloc` must be called if solution sensitivities are desired. This routine initializes and allocates memory for forward sensitivity calculations. At this stage the user specifies the number of sensitivities to be computed, the forward sensitvity method (simultaneous corrector or staggered corrector), the model parameters, as well as the initial values for the sensitivity variables.

(9) `CVode` solves the problem. The solver routine is typically called in a loop over the desired output times. The user can have the solver take internal steps until it has reached the user-specified t_{out} or return control to the user's main program after taking one successful step. Additionally, the user can direct the solver to test t_{stop} so that the integration never proceeds beyond this value.

(10) `CVodeGetSens` extracts the sensitivity solution vectors. If forward sensitivity analysis had been enabled in step 8, solutions sensitivities are computed at the same time as the ODE solution and are available to the user through this routine.

(11) `CVodeGet*` Optional outputs and statistics for the main solver are available through extraction functions. A complete list of the optional outputs from CVODES is given in Table II.

(12) `CVodeSensGet*` Optional outputs and statistics related to the solution of the sensitivity systems are available through additional extraction functions (see Table II).

(13) `CVDenseGet*`, `CVBandGet*`, `CVDiagGet*` or `CVSpgmrGet*` Optional statistics from the linear solver module can be obtained through some of the routines given in Table II.

(14) `CVodeFree` and the vector specification destructor. To complete the process, the user must make the appropriate calls to free memory that was allocated in the previous steps (vector specification objects, solver memory block, and any user data).

If there are any quadrature equations that must also be integrated, the user's main program must construct an additional NVECTOR specification object. Integration of the quadrature variables is activated and initialized through a call to the CVODES routine `CVodeQuadMalloc`, which must specify the user-provided routine for the evaluation of the quadrature integrands and the integration tolerances for quadrature variables. As before, the user has the option of changing from their default values various quantities controlling the quadrature integration (see Table I). All these calls must preceed any call to the main CVODES solver routine. After a successful return from `CVode`, the quadrature variables are accessible through a call to `CVodeGetQuad`, and solver statistics related to quadrature integration are available through the routines listed in Table II.

Adjoint sensitivity analysis inherently affects to a much greater extent the user interface, mainly due to the coupling between the forward and backward integration phases. In designing the user interface to the adjoint sensitivity module in CVODES we have strived to maintain the same "look and feel" as for that used for ODE and forward sensitivity solution. The initialization and set-up of the forward phase

Table I. Optional inputs for CVODES, CVSDENSE, CVSBAND, and CVSSPGMR

Optional input	Routine name	Default
CVODES solver		
Data for right-hand side routine	`CVodeSetFdata`	NULL
Pointer to an error file	`CVodeSetErrFile`	NULL
Maximum order for BDF method	`CVodeSetMaxOrd`	5
Maximum order for Adams method	`CVodeSetMaxOrd`	12
Maximum no. of internal steps before t_{out}	`CVodeSetMaxNumSteps`	500
Maximum no. of warnings for $h < U$	`CVodeSetMaxHnilWarns`	10
Flag to activate stability limit detection	`CVodeSetStabLimDet`	FALSE
Initial step size	`CVodeSetInitStep`	estimated
Minimum absolute step size	`CVodeSetMinStep`	0.0
Maximum absolute step size	`CVodeSetMaxStep`	∞
Value of t_{stop}	`CVodeSetStopTime`	∞
Maximum no. of error test failures	`CVodeSetMaxErrTestFails`	7
Maximum no. of nonlinear iterations	`CVodeSetMaxNonlinIters`	3
Maximum no. of convergence failures	`CVodeSetMaxConvFails`	10
Coefficient in the nonlinear convergence test	`CVodeSetNonlinConvCoef`	0.1
Error control on quadrature variables	`CVodeSetQuadErrCon`	FULL
Data for quadrature right-hand side routine	`CVodeSetQuadFdata`	NULL
Sensitivity right-hand side routine	`CVodeSetSensRhsFn`	internal DQ
Data for sensitivity right-hand side routine	`CVodeSetSensFdata`	NULL
Error control on sensitivity variables	`CVodeSetSensErrCon`	FULL
Control for difference quotient approximation	`CVodeSetSensRho`	0.0
Vector of problem parameter scalings	`CVodeSetSensPbar`	NULL
Relative tolerance for sensitivity variables	`CVodeSetSensReltol`	estimated
Absolute tolerance for sensitivity variables	`CVodeSetSensAbstol`	estimated
CVSDENSE linear solver		
Dense Jacobian routine	`CVDenseSetJacFn`	internal DQ
Data for Jacobian routine	`CVDenseSetJacData`	NULL
CVSBAND linear solver		
Band Jacobian routine	`CVBandSetJacFn`	internal DQ
Data for Jacobian routine	`CVBandSetJacData`	NULL
CVSSPGMR linear solver		
Type of Gram-Schmidt orthogonalization	`CVSpgmrSetGSType`	classical GS
Ratio between linear and nonlinear tolerances	`CVSpgmrSetDelt`	0.05
Preconditioner setup routine	`CVSpgmrSetPrecSetupFn`	NULL
Preconditioner solve routine	`CVSpgmrSetPrecSolveFn`	NULL
Data for preconditioner routines	`CVSpgmrSetPrecData`	NULL
Jacobian times vector routine	`CVSpgmrSetJacTimesVecFn`	NULL
Data for Jacobian times vector routine	`CVSpgmrSetJacData`	NULL

is the same as above. Before calling the main solver for the forward integration, the user must call the CVODES routine `CVadjMalloc` to initialize and allocate memory for the structure holding the check-pointing and interpolation data. The forward integration and check-point generation is done through a call to `CVodeF`, a wrapper around the `CVode` routine in step 9 above. The initialization, set-up, and solution of the adjoint problem is then done in the same way as for a regular forward ODE integration but calling CVODES and linear solver wrapper routines that have the names mentioned before with the suffix `B` attached. Some examples of such CVODES routines are: `CVodeCreateB`, `CVodeSpgmrB`, `CVodeMallocB`, and `CVodeB`.

Table II. Principal optional outputs from CVODES. Additional optional statistics (not listed) are available for the staggered corrector forward sensitivity method.

Optional output	**Routine name**
Size of CVODES integer workspace	`CVodeGetIntWorkSpace`
Size of CVODES real workspace	`CVodeGetRealWorkSpace`
Cumulative number of internal steps	`CVodeGetNumSteps`
No. of calls to r.h.s. function	`CVodeGetNumRhsEvals`
No. of calls to linear solver setup routine	`CVodeGetNumLinSolvSetups`
No. of local error test failures that have occurred	`CVodeGetNumErrTestFails`
Order used during the last step	`CVodeGetLastOrder`
Order to be attempted on the next step	`CVodeGetCurrentOrder`
Order reductions due to stability limit detection	`CVodeGetNumStabLimOrderReds`
Actual initial step size used	`CVodeGetActualInitStep`
Step size used for the last step	`CVodeGetLastStep`
Step size to be attempted on the next step	`CVodeGetCurrentStep`
Current internal time reached by the solver	`CVodeGetCurrentTime`
Suggested factor for tolerance scaling	`CVodeGetTolScaleFactor`
Error weight vector for state variables	`CVodeGetErrWeights`
Estimated local error vector	`CVodeGetEstLocalErrors`
No. of nonlinear solver iterations	`CVodeGetNumNonlinSolvIters`
No. of nonlinear convergence failures	`CVodeGetNumNonlinSolvConvFails`
No. of calls to quadrature r.h.s. function	`CVodeGetNumQuadRhsEvals`
No. of quadrature local error test failures	`CVodeGetNumQuadErrTestFails`
Error weight vector for quadrature variables	`CVodeGetQuadErrWeights`
No. of calls to sensitivity r.h.s. function	`CVodeGetNumSensRhsEvals`
No. of calls to r.h.s. function due to (11) or (12)	`CVodeGetNumRhsEvalsSens`
No. of sensitivity local error test failures	`CVodeGetNumSensErrTestFails`
No. of calls to linear solver setup for forward SA	`CVodeGetNumSensLinSolvSetups`
Error weight vectors for sensitivity variables	`CVodeGetSensErrWeights`
No. of nonlinear solver iterations for forward SA	`CVodeGetNumSensNonlinSolvIters`
No. of sensitivity nonlinear convergence failures	`CVodeGetNumSensNonlinSolvConvFails`

5. AVAILABILITY

The CVODES package has been released under a BSD open source license and is freely available at the web site **www.llnl.gov/CASC/sundials**, or through the DOE ACTS web site at **acts.nersc.gov/sundials/main.html**.

6. CONCLUSIONS

CVODES is the first in a series of new additions to SUNDIALS. The new codes, IDAS and KINSOLS, together with CVODES, will provide sensitivity analysis for all the classes of problems addressed by the basic SUNDIALS solvers. These new capabilities extend the versatility and functionality of the SUNDIALS solvers in addressing new classes of applications, such as dynamically-constrained optimization, inversion, and uncertainty quantification.

Like all of SUNDIALS, CVODES is under active development. An area of particular interest is in the automatic generation of the sensitivity equations. A parser and code generator for the automatic generation of derivative approximations using the complex step method is underway. Automatic differentiation (AD) tools will be incorporated as they become available; we are especially interested in adding reverse AD capabilities to the SUNDIALS adjoint sensitivity solvers. Finally, to address language interoperability issues and thus facilitate the use of the SUNDIALS

solvers for users of other programming languages, we plan to generate Babel [Kohn et al. 2001] wrappers for them.

REFERENCES

Brown, P. N., Byrne, G. D., and Hindmarsh, A. C. 1989. VODE, a Variable-Coefficient ODE Solver. *SIAM J. Sci. Stat. Comput. 10*, 1038–1051.

Brown, P. N. and Hindmarsh, A. C. 1989. Reduced Storage Matrix Methods in Stiff ODE Systems. *J. Appl. Math. & Comp. 31*, 49–91.

Byrne, G. D. 1992. Pragmatic Experiments with Krylov Methods in the Stiff ODE Setting. In *Computational Ordinary Differential Equations*, J. Cash and I. Gladwell, Eds. Oxford University Press, Oxford, 323–356.

Byrne, G. D. and Hindmarsh, A. C. 1999. PVODE, An ODE Solver for Parallel Computers. *Intl. J. High Perf. Comput. Apps. 13(4)*, 254–365.

Cao, Y., Li, S., Petzold, L. R., and Serban, R. 2003. Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and its Numerical Solution. *SIAM J. Sci. Comput. 24(3)*, 1076–1089.

Caracotsios, M. and Stewart, W. E. 1985. Sensitivity Analysis of Initial Value Problems with Mixed ODEs and Algebraic Equations. *Computers and Chemical Engineering 9*, 359–365.

Cohen, S. D. and Hindmarsh, A. C. 1996. CVODE, a Stiff/Nonstiff ODE Solver in C. *Computers in Physics 10(2)*, 138–143.

Feehery, W. F., Tolsma, J. E., and Barton, P. I. 1997. Efficient Sensitivity Analysis of Large-Scale Differential-Algebraic Systems. *Applied Numer. Math. 25(1)*, 41–54.

Hindmarsh, A. C., Brow, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. 2003. SUNDIALS, Suite of Nonlinear and Differential/Algebraic Equation Solvers. Submitted.

Hindmarsh, A. C. and Serban, R. 2002. User Documentation for CVODES, An ODE Solver with Sensitivity Analysis Capabilities. Tech. Rep. UCRL-MA-148813, LLNL. July.

Kohn, S., Kumfert, G., Painter, J., and Ribbens, C. 2001. Divorcing Language Dependencies from a Scientific Software Library. In *10th SIAM Conference on Parallel Processing*. Portsmouth, VA.

Maly, T. and Petzold, L. R. 1997. Numerical Methods and Software for Sensitivity Analysis of Differential-Algebraic Systems. *Applied Numerical Mathematics 20*, 57–79.

Radhakrishnan, K. and Hindmarsh, A. C. 1994. Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations. Tech. Rep. UCRL-ID-113855, LLNL. march.

Received Month Year; revised Month Year; accepted Month Year

University of California
Lawrence Livermore National Laboratory
Technical Information Department
Livermore, CA 94551