Proceedings of IDETC/CIE 2005
ASME International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
September 24-28, 2005, Long Beach, California, USA

DETC2005-85597

CVODES, THE SENSITIVITY-ENABLED ODE SOLVER IN SUNDIALS*

Radu Serban†
Alan C. Hindmash
Center for Applied Scienti£c Computing
Lawrence Livermore National Laboratory
P.O. Box 808, L-560, Livermore, California, 94551
Email: radu@llnl.gov, alanh@llnl.gov

ABSTRACT

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 ¤avors of staggered corrector methods. Its adjoint sensitivity module provides a combination of checkpointing and cubic Hermite interpolation for the ef£cient generation of the forward solution during the adjoint system integration.

We describe the current capabilities of CVODES, its design principles, and its user interface, and provide an example problem to illustrate the performance of CVODES.

*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.

†Address all correspondence to this author.

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 [1] and VODPK [2]. The capabilities of both VODE and VODPK have been combined in the C-language packages CVODE [3] and PVODE [4], later merged under the suite SUNDIALS [5] 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 quanti£es 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. SUNDIALS is currently being expanded to include sensitivity-capable variants of all its basic solvers. The £rst one, CVODES, released in July 2002, is written with a functionality and user interface that is a superset of that of CVODE. In that sense, CVODES is backward compatible with 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* si-

multaneously 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 £nal-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 £le. This feature is essential in certain sensitivity analysis computations and impossible in Fortran.

Like all the SUNDIALS solvers, CVODES is written in ANSI-C. Among the advantages of using C, we mention portability of the solver libraries, compiler availability, a standard dynamic memory allocation mechanism, and the ability to de£ne complex data structures. The design of the SUNDIALS solvers does not impede interlanguage operability. As an example of using a SUNDIALS solver with Fortran user code, the reader is refered to the Fortran-C interface provided for CVODE [5, 6].

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. Section 5 presents an example problem and its solution on a parallel machine. We conclude with indications on software availability in Section 6 and with some £nal remarks and directions of current and future development in Section 7.

2 Algorithms

CVODES solves initial value problems with free parameters. Such problems can be stated as

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

where $y \in \mathbf{R}^N$ is the vector of state variables, $\dot{y} = dy/dt$, and $p \in \mathbf{R}^{N_p}$ are problem parameters. Additionally, CVODES can also compute £rst order derivative information, performing either *forward sensitivity analysis* or *adjoint sensitivity analysis*. In the £rst 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 its main integration algorithm with CVODE, the interested reader is directed to [5].

2.1 ODE Integration

CVODES solves the IVP using either Backward Differentiation Formula (BDF) methods or Adams-Moulton methods. Both are implemented with dynamically varying stepsize and order, based on the control of local errors to meet user-speci£ed tolerances. A central feature of the method is the solution, at each time step, of a nonlinear system of size N, of the form

$$F(y_n) \equiv y_n - \gamma_n f(t_n, y_n, p) - a_n = 0 \tag{2}$$

where γ_n is a scalar and a_n is a constant vector. This system is solved by either functional (£xpoint) iteration or some form of Newton iteration. In the latter case, the matrix in the linear system for Newton corrections has the form $M = I - \gamma_n J_n$, where $J_n = \partial f/\partial y$ at (t_n, y_n).

CVODES also incorporates an algorithm for special treatment of quadratures depending on the solution y of (1). An ef£cient quadrature computation is needed in the context of adjoint sensitivity analysis (see Section 2.3). Evaluation of integrals of the form $G = \int_{t_0}^{t_f} g(t, y, p)dt$ can be done ef£ciently using the underlying linear multistep method interpolating polynomials by appending to (1) an additional ODE $\dot{\phi} = g(t, y, p)$, with initial condition $\phi(t_0) = 0$, in which case $G = \phi(t_f)$. In the context of an implicit ODE integrator, since the right-hand side of this additional equation does not depend on ϕ, such equations need not participate in the solution of the nonlinear system (2). CVODES allows the user to identify these equations separately from those in (1) and provides the option of including or excluding ϕ from the error control algorithm. A similar treatment of quadratures is included in the DASPK3 code [7, 8].

A complete description of the CVODES integration algorithm, including the nonlinear solver convergence, error control mechanism, and heuristics related to stopping criteria and £nite-difference parameter selection, is given in [5].

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 (1). 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 in¤uential 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 de£ned as the vector $s_i(t) = \partial y(t)/\partial p_i$ and satis£es the following *forward sensitivity equations* (or in short *sensitivity equations*):

$$\dot{s}_i = \frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial p_i}, \quad s_i(t_0) = \frac{\partial y_0(p)}{\partial p_i}, \tag{3}$$

obtained by applying the chain rule of differentiation to the original ODEs (1).

When performing forward sensitivity analysis, CVODES carries out the time integration of the combined system, (1) and (3), 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 ($N_s \leq N_p$). 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 brie¤y describe three methods that have been proposed for the solution of the combined ODE and sensitivity system for the vector $\hat{y} = [y, s_1, \ldots, s_{N_s}]$.

Staggered Direct. In this approach [9], the nonlinear system (2) is £rst solved and, once an acceptable numerical solution is obtained, the sensitivity variables at the new step are found by directly solving (3) after the BDF discretization is used to eliminate $\dot{s}$. Although the system matrix of the above linear system is based on exactly the same information as the matrix M, it must be updated and factored at every step of the integration, in contrast to M which is updated only ocasionally. For problems with many parameters (relative to the problem size), the staggered direct method can outperform the methods described below [10]. However, the computational cost associated with matrix updates and factorizations makes this method unattractive for problems with many more states than parameters (such as those arising from semidiscretization of PDEs).

Simultaneous Corrector. In this method [11], the BDF discretization is applied simultaneously to both the original equations (1) and the sensitivity systems (3) resulting in an extended nonlinear system in the unknown $\hat{y} = [y, s_1, \ldots, s_{N_s}]$:

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

where $\hat{f} = [f, \ldots, (\partial f/\partial y) s_i + (\partial f/\partial p_i), \ldots]$ 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 using either functional or Newton iteration. In the latter case, Maly and Petzold have shown that 2-step quadratic convergence can be attained with a modi£ed Newton scheme which uses only the block-diagonal portion of the iteration matrix. 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 to update the sensitivity portions of the residual.

Staggered corrector. In this approach [12], as in the staggered direct method, the nonlinear system (2) is solved £rst for the state variables. Then, a separate nonlinear iteration is used to solve the sensitivity system. In this approach, the vectors $\partial f/\partial p_i$ need be updated only once per integration step, after the state correction phase has converged. When using functional iteration, this amounts to using a stationary iterative method for the linear system (3). When using Newton iteration, this amounts to using a modi£ed-Newton iteration to solve the linear system (3), the Newton iteration matrix being only an approximation to the system matrix. An important observation is that the staggered corrector method, combined with Newton iterations and the SPGMR linear solver, effectively results in a staggered direct method [13]. Indeed, SPGMR requires only the action of the matrix M on a vector and this can be provided with up-to-date Jacobian information. Therefore, the modi£ed Newton procedure will theoretically converge after one iteration.

CVODES implements the simultaneous corrector method and two ¤avors 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 £rst variant of the staggered corrector method requires the convergence of the nonlinear sensitivity iterations 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 ($i = 1, \ldots, N_s$), 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.

We note that the DASPK3.0 code [7, 14] implements the staggered direct, simultaneous corrector, and staggered corrector methods. The code DSL48S [12, 15] also contains the staggered corrector method.

2.2.2 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 Sensitivity right-hand side. There are several methods for evaluating the right-hand side of the sensitivity systems (3): analytic evaluation, automatic differentiation, complex-step approximation, and £nite differences (or directional derivatives).

Since it allows for user-de£ned functions for the evaluation of any and all derivative information, CVODES provides all the software hooks for implementing interfaces to automatic differentiation or complex-step approximation. We have prototyped an automated code generator tool (not included in the CVODES distribution) which parses C code and generates C++ code to perform complex arithmetic. The user's right-hand side function can thus be transformed to allow for the automatic generation of derivative information using complex-step approximations [16]. This approach allows for very accurate numerical estimation of derivatives as it circumvents the subtraction cancellation error typical for £nite difference methods. However, any code transformation approach to the automatic generation of derivative information from C functions (for example ADIC [17]) has the disadvantage of requiring transformations on the user's data structures, which are otherwise treated as black boxes by CVODES. This makes the design of general interfaces a challenging task, but we are investigating avenues to overcome this issue.

The default option in CVODES for the evaluation of sensitivity right-hand sides is to use £nite difference-based approximations for the terms $(\partial f/\partial y)s_i$ and $\partial f/\partial p_i$, or directional derivatives to evaluate $(\partial f/\partial y)s_i + \partial f/\partial p_i$. As is typical for £nite 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 £nite 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 (3) can be evaluated separately:

$$\frac{\partial f}{\partial y} s_i \approx \frac{f(t, y+\sigma_y s_i, p) - f(t, y-\sigma_y s_i, p)}{2\sigma_y}, \tag{4}$$

$$\frac{\partial f}{\partial p_i} \approx \frac{f(t, y, p+\sigma_i e_i) - f(t, y, p-\sigma_i e_i)}{2\sigma_i}, \tag{4'}$$

simultaneously:

$$\frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial p_i} \approx \frac{f(t, y+\sigma s_i, p+\sigma e_i) - f(t, y-\sigma s_i, p-\sigma e_i)}{2\sigma}, \tag{5}$$

or adaptively switching between (4)+(4') and (5), depending on the relative size of the estimated £nite difference increments σ_i and σ_y. These increments are

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

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 (1), we wish to evaluate the gradient dG/dp of

$$G(p) = \int_{t_0}^{t_f} g(t, y, p)\, dt, \tag{6}$$

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. The gradient of G with respect to p is simply

$$\frac{dG}{dp} = \lambda^T(t_0) s(t_0) + \int_{t_0}^{t_f} \left(\frac{\partial g}{\partial p} + \lambda^T \frac{\partial f}{\partial p} \right) dt, \tag{7}$$

where λ is solution of

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial y}\right)^T \lambda - \left(\frac{\partial g}{\partial y}\right)^T, \quad \lambda(t_f) = 0 \tag{8}$$

and $s(t_0) = \partial y_0/\partial p$. The gradient of $g(t_f, y, p)$ with respect to p can be then obtained by using the Leibnitz differentiation rule [18].

The first thing to notice about the adjoint system (8) is that there is no explicit specification of the parameters p; this implies that, once the solution λ is found, the formula (7) can then be used to find the gradient of G with respect to any of the parameters 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 (1). Therefore, a procedure is needed for providing the states y obtained during a forward integration phase of (1) to CVODES during the backward integration phase of (8). The approach adopted in CVODES, similar to that implemented in DASPKADJOINT [8], is justified below.

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 check-pointing scheme which, at the cost of at most one additional forward integration, offers the best possible estimate of memory requirements for adjoint sensitivity analysis. Note that truly optimal checkpointing [19] cannot be used since the number of integration steps is not known apriori.

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. The backward integration from check point $i+1$ to check point i is preceded 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 approach, illustrated in Fig. 1, 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.

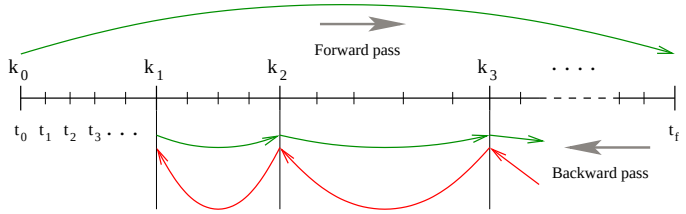

Figure 1. CHECK-POINTING ALGORITHM FOR GENERATION OF THE FORWARD SOLUTION DURING THE BACKWARD INTEGRATION PHASE.

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 (1), including the adjoint system (8), as well as any other quadrature ODEs that may be needed in evaluating the integral in (7). In particular, for ODE systems arising from semi-discretization of time-dependent PDEs, this feature allows for integration either of the discretized adjoint PDE system or of the adjoint of the discretized PDE, since these two formulations are not equivalent [20, 21].

Finally, we mention that, when using the backward integration module for adjoint sensitivity analysis, the CVODES interface allows for user-provided functions for the evaluation of the adjoint systems that are generated through reverse automatic differentiation. Due to the current development stage of reverse AD tools for C codes, CVODES cannot provide generic wrappers (as done, for example, in DASPKADJOINT for Fortran77 codes). At this time, the burden of interfacing CVODE with AD-generated functions must rely on the user.

3 Code Organization

As mentioned before, the SUNDIALS suite consists of the basic solvers CVODE (for ODE systems), KINSOL (for nonlinear algebraic systems), and IDA (for DAE systems) and of sensitivity-capable variants, CVODES, IDAS, and KINSOLS (the last two being currently under development). The overall organization of the CVODES package, as well as its relationship to SUNDIALS, is shown in Fig. 2. The basic elements of the CVODES 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. Modules which are shared across the entire SUNDIALS suite include generic linear system solvers, and the NVECTOR modules (described further below).

The central CVODES 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

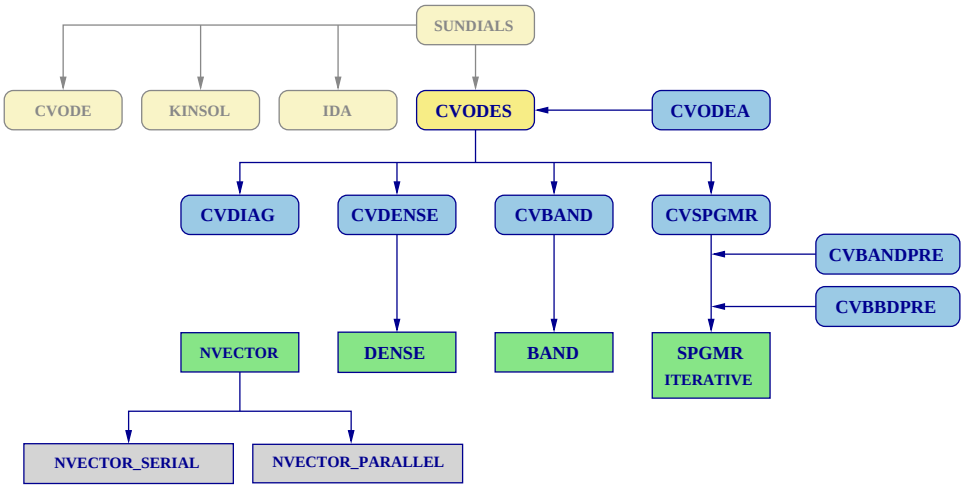

Figure 2. OVERALL STRUCTURE DIAGRAM OF TE CVODES PACAGE. MODULES SPECIFIC TO CVODES ARE DISTINGUISHED BY ROUNDED BOXES, WHILE GENERIC SOLVER AND AUXILIARY MODULES ARE IN SQUARE BOXES.

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 speci£ed 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 integration 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 CVODES package includes four linear system solution modules, of which three use direct methods, and one uses scaled preconditioned GMRES, a Krylov subpsace method. For the latter, two preconditioner modules are also included, one for use on serial computers, and one for parallel. All of these are virtually identical to the corresponding modules for CVODE, which are described in detail in [5]. In addition, the user of CVODES may supply his/her own linear solver module, following speci£cations given in the user documentation. Thus an existing linear system solver can be incorporated by providing short interface functions between CVODES and the linear system solver.

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 speci£c 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 functions for CVODES. At a minimum, the user must provide a function for the evaluation of the ODE right-hand side and, if performing adjoint sensitivity analysis, a function for the evaluation of the right-hand side of the adjoint system. Optional user-provided functions 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 function 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 functions.

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 de£nes an NVECTOR structure speci£cation, 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 speci£c NVECTOR implementation, CVODES is thus independent of a speci£c 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 ef£cient 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 functions for operations on N-vectors, such as creation, destruction, summation, and dot product; (2) a function to construct an NVECTOR speci£cation structure for this particular implementation, which de£nes 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 speci£cation 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 [22].

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 functions, 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 con£guring 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 speci£cation constructor must £rst be called. For the two NVECTOR implementation provided with SUNDIALS, serial and MPI parallel, the constructor functions are `NV_New_Serial` and `NV_New_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*` functions can now be used to change various controls from their default values. Choices and default values are given in Table 2.
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 function 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 linear solver object (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 3.
7. `CVodeSetSens*` functions can be called to change from their default values the optional inputs that control the integration of the sensitivity systems (see Table 2).
8. `CVodeSensMalloc` must be called if solution sensitivities are desired. This function initializes and allocates memory for forward sensitivity calculations. At this stage the user speci£es the number of sensitivities to be computed, the forward sensitivity method the model parameters, as well as the initial values for the sensitivity variables.
9. `CVode` solves the problem. The solver function 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-speci£ed 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 function.
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 4.

12. `CVodeGet*Sens*`. Optional outputs related to the solution of the sensitivity systems are available through additional extraction functions (see Table 4). More optional statistics (not listed) are available for the staggered corrector forward sensitivity method.
13. `CVDenseGet*`, `CVBandGet*`, `CVDiagGet*` or `CVSpgmrGet*`. These functions provide optional statistics from the linear solver module.
14. `CVodeFree`. To complete the process, the user must make the appropriate calls to free memory that was allocated in the previous steps (vector speci£cation 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 speci£cation object. Integration of the quadrature variables is activated and initialized through a call to the CVODES function `CVodeQuadMalloc`, which must specify the user-provided function 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 2). All these calls must preceed any call to the main CVODES solver function. 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 functions listed in Table 4.

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 is the same as above. Before calling the main solver for the forward integration, the user must call the CVODES function `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` function 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 functions that have the names mentioned before with the suf£x `B` attached. Some examples of such CVODES functions are: `CVodeCreateB`, `CVSpgmrB`, `CVodeMallocB`, and `CVodeB`.

5 Parallel Example Problem

The most preeminent advantage of CVODES over existing sensitivity solvers is the possibility of solving very large-scale problems on massively parallel computers. To illustrate this point we present speedup results for the integration and forward sensitivity analysis for an ODE system generated from the following 2-species diurnal kinetics advection-diffusion PDE system in 2 space dimensions:

$$\frac{dc_i}{dt} = K_h \frac{d^2c_i}{dx^2} + v\frac{dc_i}{dx} + K_v \frac{d^2c_i}{dz^2} + R_i(c_1, c_2, t), \quad \text{for } i = 1, 2, \tag{9}$$

where

$$\begin{aligned} R_1(c_1, c_2, t) &= -q_1 c_1 c_3 - q_2 c_1 c_2 + 2q_3(t) c_3 + q_4(t) c_2, \\ R_2(c_1, c_2, t) &= q_1 c_1 c_3 - q_2 c_1 c_2 - q_4(t) c_2, \end{aligned} \tag{10}$$

K_h, K_v, v, q_1, q_2, and c_3 are constants, and $q_3(t)$ and $q_4(t)$ vary diurnally. The problem is posed on the square $0 \le x \le 20$, $30 \le z \le 50$ (all in km), with homogeneous Neumann boundary conditions, and for time t in $0 \le t \le 86400$ (1 day). The PDE system is treated by central differences on a uniform mesh, except for the advection term, which is treated with a biased 3-point difference formula. The initial pro£les are proportional to a simple polynomial in x and a hyperbolic tangent function in z.

The solution with CVODES is done with the BDF/GMRES method (i.e. using the CVSPGMR linear solver) and the block-diagonal part of the Newton matrix as a left preconditioner. A copy of the block-diagonal part of the Jacobian is saved and conditionally reused within the preconditioner setup function.

The problem is solved by CVODES on P processors, treated as a rectangular process grid of size $p_x \times p_z$. Each processor contains a subgrid of size $n = n_x \times n_z$ of the (x,z) mesh. Thus the actual mesh size is $N_x \times N_z = (p_x n_x) \times (p_z n_z)$, and the ODE system size is $N = 2N_x N_z$. Parallel performance tests were performed on ASCI Frost, a 68-node, 16-way SMP system with POWER3 375 MHz processors and 16 GB of memory per node. Speedup results for a global problem size of $N = 2N_x N_y = 2 \cdot 1600 \cdot 400 = 1280000$ shown in Fig. 3 and listed in Table 1. We present timing results for the integration of only the state equations (column STATES), as well as for the computation of forward sensitivities with respect to the diffusion coef£cients K_h and K_v using the staggered corrector method without and with error control on the sensitivity variables (columns STG and STG_FULL, respectively). We note that there was not enough memory to solve the problem (even without carrying sensitivities) on fewer processors.

The departure from the ideal line of slope -1 is explained by the interplay of several con¤icting processes. On one hand, when increasing the number of processors, P, the preconditioner quality decreases, as it incorporates a smaller and smaller fraction of the Jacobian and the cost of inter-process communication increases. On the other hand, decreasing P leads to an increase

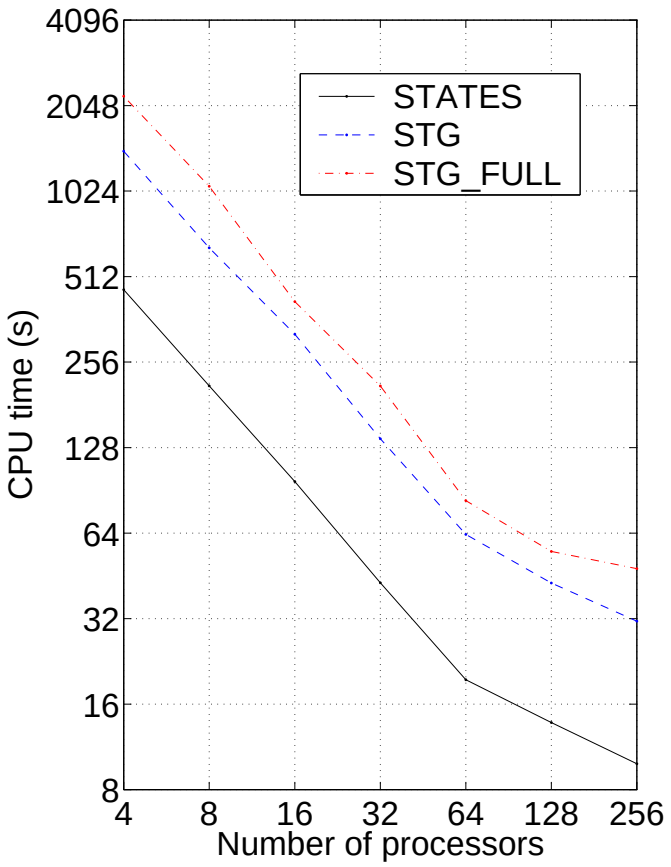

Figure 3. SPEEDUP RESULTS FOR THE INEGRATION OF THE STATE EQUATIONS ONLY (SOLID LINE AND COLUMN 'STATES'), STAGGERED SA WITHOUT ERROR CONTROL ON THE SENSITIVITY VARIABLES (DASHED LINE AND COLUMN 'STG'), AND STAGGERED SA WITH FULL ERROR CONTROL (DOTTED LINE AND COLUMN 'STG_FULL').

in the cost of the preconditioner setup phase and to a larger local problem size which can lead to a point where a node starts memory paging to disk.

6 Availability

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

7 Conclusions

CVODES is the £rst 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 quanti£cation.

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. We are currently investigating alternatives to checkpointing within the adjoint solver in CVODES: one direction is in using reduced order models of the forward problem, while another is in storing the complete decision history on the £rst forward pass and re-using it on the second pass. 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 [23] wrappers for them.

Table 1. SPEEDUP RESULTS

P	STATES	STG	STG_FULL
4	460.31	1414.53	2208.14
8	211.20	646.59	1064.94
16	97.16	320.78	417.95
32	42.78	137.51	210.84
64	19.50	63.34	83.24
128	13.78	42.71	55.17
256	9.87	31.33	47.95

REFERENCES

[1] Brown, P. N., Byrne, G. D., and Hindmarsh, A. C., 1989. "VODE, a variable-coef£cient ODE solver". *SIAM J. Sci. Stat. Comput.*, **10**, pp. 1038–1051.

[2] 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, pp. 323–356.

[3] Cohen, S. D., and Hindmarsh, A. C., 1996. "CVODE, a stiff/nonstiff ODE solver in C". *Computers in Physics*, **10**(2), pp. 138–143.

[4] Byrne, G. D., and Hindmarsh, A. C., 1999. "PVODE, an

ODE solver for parallel computers". *Intl. J. High Perf. Comput. Apps.*, **13**(4), pp. 254–365.

[5] Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S., 2004. "SUNDIALS, Suite of Nonlinear and Differential/Algebraic Equation Solvers". *ACM Trans. Math. Softw.*(in press).

[6] Hindmarsh, A. C., and Serban, R., 2004. User documentation for CVODE v2.2.0. Tech. Rep. UCRL-SM-208108, LLNL.

[7] Li, S., and Petzold, L. R., 2000. "Software and algorithms for sensitivity analysis of large-scale differential-algebraic systems". *J. Comp. Appl. Math.*, **125**, pp. 131–145.

[8] Li, S., and Petzold, L. R., 2000. Description of DASPKADJOINT: An adjoint sensitivity solver for differential-algebraic equations. Tech. rep., Dept. of Computer Science, UCSB.

[9] Caracotsios, M., and Stewart, W. E., 1985. "Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations". *Computers and Chemical Engineering*, **9**, pp. 359–365.

[10] Li, S., Petzold, L. R., and Zhu, W., 1999. "Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem". *Appl. Num. Math.*, **32**(2), pp. 161–174.

[11] Maly, T., and Petzold, L. R., 1996. "Numerical methods and software for sensitivity analysis of differential-algebraic systems". *Appl. Num. Math.*, **20**, pp. 57–79.

[12] Feehery, W. F., Tolsma, J. E., and Barton, P. I., 1997. "Ef£cient sensitivity analysis of large-scale differential-algebraic systems". *Appl. Num. Math.*, **25**(1), pp. 41–54.

[13] Tocci, M. D., 2001. "Sensitivity analysis of large-scale time dependent PDEs". *Appl. Num. Math.*, **37**(1), pp. 109–125.

[14] Li, S., and Petzold, L. R., 1999. Design of new DASPK for sensitivity analysis. Tech. rep., Dept. of Computer Science, UCSB.

[15] Feehery, W. F., 1998. Dsl48s manual. Tech. Rep. ABACUSS Project Report 98/1, MIT.

[16] Martins, J., Sturdza, P., and Alonso, J., 2003. "The complex-step derivative approximation". *ACM Trans. Math. Softw.*, **29**(3), pp. 245–262.

[17] Bischof, C. H., Roh, L., and Mauer-Oats, A. G., 1997. "ADIC: An extensible automatic differentiation tool for ANSI-C". *Software - Practive and Experience*, **27**(12), pp. 1427–1456.

[18] 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), pp. 1076–1089.

[19] Griewank, A., and Walther, A., 2000. "Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of cimputatinal differentiation". *ACM Trans. Math. Softw.*, **26**(1), pp. 19–45.

[20] Arian, E., and Salas, M., 1997. Admitting the inadmissible: Adjoint formulation for incomplete cost functionals in aerodynamic optimization. Tech. Rep. 97-69, ICASE.

[21] Li, S., and Petzold, L. R., 2004. "Adjoint sensitivity analysis for time-dependent partial differential equations with adaptive mesh re£nement". *J. Comp. Phys.*, **198**(1), pp. 310–325.

[22] Hindmarsh, A. C., and Serban, R., 2004. User documentation for CVODES v2.1.0. Tech. Rep. UCRL-SM-208111, LLNL.

[23] Kohn, S., Kumfert, G., Painter, J., and Ribbens, C., 2001. "Divorcing language dependencies from a scienti£c software library". In 10th SIAM Conference on Parallel Processing.

Table 2. OPTIONAL INPUTS FOR THE MAIN CVODES SOLVER

Optional input	**Function name**	**Default**
Pointer to an error output £le	`CVodeSetErrFile`	NULL
Data for right-hand side function	`CVodeSetFdata`	NULL
Maximum order for BDF (or Adams) method	`CVodeSetMaxOrd`	5 (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
Coef£cient in the nonlinear convergence test	`CVodeSetNonlinConvCoef`	0.1
Data for quadrature right-hand side function	`CVodeSetQuadFdata`	NULL
Error control on quadrature variables	`CVodeSetQuadErrCon`	FALSE
Tolerances for quadrature variables	`CVodeSetQuadTolerances`	none
Sensitivity right-hand side function	`CVodeSetSensRhsFn`	internal DQ
Data for sensitivity right-hand side function	`CVodeSetSensFdata`	NULL
Error control on sensitivity variables	`CVodeSetSensErrCon`	FALSE
Control for difference quotient approximation	`CVodeSetSensRho`	0.0
Vector of problem parameter scalings	`CVodeSetSensPbar`	NULL
Tolerances for sensitivity variables	`CVodeSetSensTolerances`	estimated

Table 3. OPTIONAL INPUTS FOR THE CVODES LINEAR SOLVER MODULES

Linear solver	**Optional input**	**Function name**	**Default**
CVDENSE	Dense Jacobian function	`CVDenseSetJacFn`	internal DQ
	Data for Jacobian function	`CVDenseSetJacData`	NULL
CVBAND	Band Jacobian function	`CVBandSetJacFn`	internal DQ
	Data for Jacobian function	`CVBandSetJacData`	NULL
CVSPGMR	Preconditioner solve function	`CVSpgmrSetPrecSolveFn`	NULL
	Preconditioner setup function	`CVSpgmrSetPrecSetupFn`	NULL
	Data for preconditioner functions	`CVSpgmrSetPrecData`	NULL
	Jacobian times vector function	`CVSpgmrSetJacTimesVecFn`	NULL
	Data for Jacobian times vector function	`CVSpgmrSetJacData`	NULL
	Type of Gram-Schmidt orthogonalization	`CVSpgmrSetGSType`	classical GS
	Ratio between linear and nonlinear tolerances	`CVSpgmrSetDelt`	0.05

Table 4. PRINCIPAL CVODES OPTIONAL OUTPUTS

Optional output	**Function 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 function	`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 (4) or (5)	`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`