SIAM J. SCI. COMPUT.
Vol. 36, No. 5, pp. C504–C523

FATODE: A LIBRARY FOR FORWARD, ADJOINT, AND TANGENT LINEAR INTEGRATION OF ODES*

HONG ZHANG† AND ADRIAN SANDU†

Abstract. FATODE is a Fortran library for the integration of ordinary differential equations with direct and adjoint sensitivity analysis capabilities. The paper describes the capabilities, implementation, code organization, and usage of this package. FATODE implements four families of methods: explicit Runge–Kutta for nonstiff problems, fully implicit Runge–Kutta, singly diagonally implicit Runge–Kutta, and Rosenbrock for stiff problems. Each family contains several methods with different orders of accuracy; users can add new methods by simply providing their coefficients. For each family the forward, adjoint, and tangent linear models are implemented. General purpose solvers for dense and sparse linear algebra are used; users can easily incorporate problem-tailored linear algebra routines. The performance of the package is demonstrated on several test problems. To the best of our knowledge FATODE is the first publicly available general purpose package that offers forward and adjoint sensitivity analysis capabilities in the context of Runge–Kutta methods. A wide range of applications is expected to benefit from its use; examples include parameter estimation, data assimilation, optimal control, and uncertainty quantification.

Key words. Runge–Kutta methods, tangent linear model, adjoint model, sensitivity analysis

AMS subject classifications. 97N80, 65L99, 49Q12

DOI. 10.1137/130912335

1. Introduction.

Many dynamical systems in science and engineering are modeled by ordinary differential equations (ODEs)

$$y' = f\left(t, y; p\right), \quad t_0 \le t \le t_F, \quad y(t_0) = y_0(p). \tag{1.1}$$

Here $y(t) \in \mathbb{R}^d$ is the solution vector, y_0 is the initial condition, and $p \in \mathbb{R}^m$ is a vector of model parameters. Stiffness results from the existence of multiple dynamical scales, with the fastest characteristic times being much smaller than the time scales of interest in the simulation. It is well known that the numerical solution of stiff systems requires unconditionally stable discretizations which allow time steps that are not bounded by the fastest time scales in the system [14]. Here we assume that the system parameters p are independent of time. In the context of the ODE system (1.1), sensitivity analysis yields derivatives of the solution with respect to the initial conditions or system parameters, as follows:

$$S_\ell(t) = \frac{\partial y(t)}{\partial p_\ell}, \quad \ell = 1, \ldots, m. \tag{1.2}$$

Two main approaches are available for computing the sensitivities (1.2). The direct (or tangent linear) method is efficient when the number of parameters is smaller than the dimension of the system ($m \ll d$), while the adjoint method is efficient when

*Submitted to the journal's Software and High-Performance Computing section March 7, 2013; accepted for publication (in revised form) April 22, 2014; published electronically October 2, 2014. This work is supported by the National Science Foundation through the awards NSF DMS–0915047, NSF CCF–0916493, NSF OCI–0904397, NSF CMMI–1130667, NSF CCF–1218454, AFOSR FA9550–12–1–0293–DEF, and AFOSR 12–2640–06.
http://www.siam.org/journals/sisc/36-5/91233.html
†Computational Science Laboratory, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 (zhang@vt.edu, sandu@cs.vt.edu).

the number of parameters is larger than the dimension of the system ($m \gg d$). Furthermore, two distinct approaches can be taken for defining adjoint sensitivities; the continuous adjoint (differentiate-then-discretize) and the discrete adjoint (discretize-then-differentiate) approaches typically lead to different computational results.

Sensitivity analysis is an essential ingredient for uncertainty quantification, parameter estimation, optimization, optimal control, and construction of reduced order models. Only a few available software packages for the solution of ODEs have the capability to compute sensitivities. One of the earlier packages is Odessa [17], which performs direct sensitivity analysis. A modern package is CVODES within SUNDIALS [24] from Lawrence Livermore National Laboratory. CVODES is able to compute direct and continuous adjoint sensitivities. Both Odessa and CVODES are based on backward differentiation formulas (BDFs). A software based on explicit Runge–Kutta (ERK) discretizations is DENSERKS [2], which implements continuous adjoint sensitivity analysis for nonstiff ODEs. The Kinetic PreProcessor (KPP) [1, 21] is a widely used tool for the simulation of chemical kinetics and incorporates Runge–Kutta and Rosenbrock solvers that are endowed with tangent linear and discrete adjoint sensitivity analysis capabilities.

In this paper we present a library of explicit/implicit Runge–Kutta and Rosenbrock solvers for the simulation of nonstiff and stiff ODEs. The library, called FATODE [27, 28, 29, 30], performs forward simulations, and sensitivity analysis via the discrete adjoint and the tangent linear methods. FATODE brings new capabilities to the constellation of available sensitivity analysis software, as follows.

- FATODE is based on the KPP library of solvers. While the KPP implementation is specifically aimed at chemical kinetic systems, the FATODE implementation is general and suitable for a wide range of applications.
- FATODE is the first available general purpose software that implements a *discrete adjoint sensitivity analysis* approach. This gives gradients that are exact, within roundoff error, and therefore highly suitable for numerical optimization problems. In contradistinction, both DENSERKS and CVODES implement a continuous adjoint approach; i.e., they solve the adjoint ODEs by applying the same numerical methods as for the forward ODEs.
- FATODE is the first general purpose package that implements sensitivity analysis for stiff systems using implicit Runge–Kutta and Rosenbrock methods. In contrast, DENSERKS uses ERK methods for nonstiff problems, while CVODES is based on linear multistep methods.
- Numerical tests show that FATODE is competitive with CVODES, from a computational efficiency perspective, as both an ODE integration package as well as a forward and adjoint sensitivity solver. In addition, the FATODE library implements not one, but multiple methods, and the user has the possibility to select the best one for the application at hand.

The paper is organized as follows. The background in section 2 reviews numerical integration algorithms, with emphasis on those implemented in FATODE, and summarizes the direct and adjoint approaches to sensitivity analysis. Section 3 discusses the array of tunable parameters offered by the package and provides general guidelines to selecting the best options for the problem at hand. Section 4 presents the code structure and implementation aspects of FATODE. Numerical tests with a nonstiff system are shown in section 5, and those with a stiff system are shown in section 6. Section 7 discusses discusses several important choices of methods and parameters. Conclusions are drawn in section 8. Additional aspects are presented in the supplementary material [31] that accompanies this paper.

2. Background.

2.1. Numerical solution of ODEs. A considerable body of work has been devoted to the numerical solution of ODE systems (1.1), as presented in the monograph [13, 14]. Many classes of numerical discretizations are available, and rigorous mathematical theories have been developed to analyze their accuracy and stability properties. Numerical discretizations can be broadly classified into explicit, which compute the new solution by repeated evaluations of the right-hand side of (1.1), and implicit, which compute the new solution by solving (non)linear systems of equations at each step. When solving nonstiff systems of ODEs the step sizes are decided solely based on accuracy requirements, and explicit methods [13] are preferred due to their lower cost per step. Implicit methods [14] are employed when solving stiff systems due to their better numerical stability properties.

FATODE implements four families of methods: ERK for nonstiff problems, and Rosenbrock, fully implicit Runge–Kutta (FIRK), and singly diagonally implicit Runge–Kutta (SDIRK) for stiff problems. Forward and adjoint sensitivity solvers are also included. The implicit methods have been previously implemented in KPP [1, 21] and have proved to be very efficient for solving many stiff chemical problems including CBM-IV [10], SAPRC [5], and NASA HSRP/AESA. The implementation of the forward implicit methods is inspired by the codes associated with the monograph [14].

All methods in FATODE are one-step integration formulas that advance the numerical solution $y_n \approx y(t_n)$ to $y_{n+1} \approx y(t_{n+1})$, $t_{n+1} = t_n + h$, using

$$y_{n+1} = \Phi^n(y_n, p), \quad n = 0, \ldots, N-1\,. \tag{2.1}$$

The specific families of methods are introduced below.

Runge–Kutta methods. A general s-stage Runge–Kutta method reads [13]

$$T_i = t_n + c_j\, h\,, \quad Y_i = y_n + h \sum_{j=1}^{s} a_{i,j}\, f\left(T_j\,,\, Y_j\right)\,, \quad i = 1, \ldots, s\,, \tag{2.2a}$$

$$y_{n+1} = y_n + h \sum_{j=1}^{s} b_j\, f\left(T_j\,,\, Y_j\right), \tag{2.2b}$$

where the coefficients

$$\boldsymbol{A} = [a_{i,j}]_{1\le i,j\le s}\,, \quad \boldsymbol{b} = [b_i]_{1\le i\le s}\,, \quad \boldsymbol{c} = [c_i]_{1\le i\le s} = \boldsymbol{A}\cdot \mathbf{1}_{(s,1)} \tag{2.3}$$

define the method and determine its accuracy and stability properties. ERK methods are characterized by the coefficients $a_{i,j} = 0$ for all i and $j \ge i$. SDIRK methods are defined by (2.2) with $a_{i,i} = \gamma$ and $a_{i,j} = 0$ for all i and $j > i$. Fully implicit methods do not enjoy any special structure of the coefficient matrix and require coupled solutions for all the stage vectors, i.e., solutions of nonlinear systems of dimension $sd \times sd$.

Rosenbrock methods. An s-stage Rosenbrock method [13] is given by the formulas

$$T_i = t_n + \alpha_i h\,, \quad Y_i = y_n + \sum_{j=1}^{i-1} \alpha_{i,j}\, k_j\,, \tag{2.4a}$$

$$k_i = h\, f\left(T_i, Y_i\right) + h\boldsymbol{f_y}(t_n, y_n)\cdot \sum_{j=1}^{i} \gamma_{i,j} k_j + \gamma_i h^2 f_t(t_n, y_n), \quad i = 1, \ldots, s\,, \tag{2.4b}$$

$$y_{n+1} = y_n + \sum_{j=1}^{s} b_j k_j \,, \tag{2.4c}$$

where particular methods are defined by their coefficients

$$\boldsymbol{\alpha} = [\alpha_{i,j}]_{1\le i,j\le s} \,, \quad \boldsymbol{b} = [b_i]_{1\le i\le s} \,, \quad \boldsymbol{\gamma} = [\gamma_{i,j}]_{1\le i,j\le s} \,, \tag{2.5}$$
$$\alpha_i = \sum_{j=1}^{i-1} \alpha_{i,j} \,, \quad \gamma_i = \sum_{j=1}^{i} \gamma_{i,j} \,; \quad \alpha_{i,j} = 0 \ \forall\, i \ge j \,; \quad \gamma_{i,j} = 0 \ \forall\, i > j \,.$$

We have $\gamma_{i,i} = \gamma$ for all i for computational efficiency. Here $\boldsymbol{f_y} = \partial f/\partial y \in \mathbb{R}^{d\times d}$ represents the Jacobian of the ODE function, and $f_t = \partial f/\partial t$, as discussed in [31, part A]. We will denote matrices and tensors by bold symbols and vectors and scalars by regular symbols. Rosenbrock methods are attractive because of their excellent stability properties and the conservation of linear invariants of the system. They typically outperform BDFs such as those implemented in SMVGEAR [9] for medium accuracy solutions.

Methods implemented in FATODE. The particular numerical schemes available in FATODE are summarized in Table 1.

The ERK methods are suitable for nonstiff systems. The orders of accuracy range between two (RK2) and eight (DOPRI-853). The cost per step is (roughly) proportional to the number of stages.

All FIRK methods have three stages and require a coupled solution of all of them. This is implemented via two LU factorizations per step, one real and one complex. Thus the FIRK cost per step is the largest among all methods in FATODE. A-stability and L-stability are linear stability properties that guarantee that truncation errors do not accumulate quickly regardless of the stepsize [14]. L-stable methods are well suited for stiff problems as they guarantee a strong damping of the fast solution transients. Stiff accuracy is a property that allows for correct solutions of nonlinear systems in the limit of infinite stiffness (when the ODE becomes a differential algebraic equation [14]).

SDIRK schemes perform a single LU factorization per time step, which is used to solve the nonlinear equations in each stage by simplified Newton equations. The number of forward-backward substitutions equals the total number of simplified Newton iteration for all stages. The methods implemented have orders two and four and are all stiffly accurate, meaning that they are suitable for stiff problems.

Rosenbrock methods have the lowest cost per time step among all implicit schemes. Only linear systems need to be solved. There is one LU factorization per time step and as many forward-backward substitutions as there are stages. All methods implemented are L-stable. The stiff accuracy property is defined differently for Rosenbrock schemes than for Runge–Kutta schemes [14], but the meaning is similar: stiffly accurate methods provide correct results in the asymptotic limit of infinite stiffness.

2.2. Sensitivity analysis. Consider an output vector $\Psi \in \mathbb{R}^o$ that depends on the solution of the system (1.1)

$$\Psi = g\left(y(t_F), p\right) + \int_{t_0}^{t_F} r\left(t, y(t), p\right)\, dt \,, \tag{2.6}$$

where $g : \mathbb{R}^{d+m} \to \mathbb{R}^o$ and $r : \mathbb{R}^{1+d+m} \to \mathbb{R}^o$. For example, the first terms can penalize the discrepancy between the model state and a target state at the final time,

TABLE 1
Time stepping methods implemented in FATODE. *(ERK, FIRK, SDIRK, and ROS stand for explicit Runge–Kutta, fully implicit Runge–Kutta, singly diagonally implicit Runge–Kutta, and Rosenbrock, respectively.)*

Family	Method	Stages	Order	Stability properties
ERK	RK2(3) [13]	3	2	Conditionally stable
	RK3(2) [13]	4	3	Conditionally stable
	RK4(3) [13]	5	4	Conditionally stable
	DOPRI5 [13]	7	5	Conditionally stable
	Verner [13]	8	6	Conditionally stable
	DOPRI853 [13]	12	8	Conditionally stable
FIRK	Radau1A [14]	3	5	L-stable
	Radau2A [14]	3	5	L-stable, stiffly accurate
	Lobatto3C [14]	3	4	L-stable, stiffly accurate
	Gauss [14]	3	6	Weakly A-stable
SDIRK	Sdirk2a	2	2	L-stable, stiffly accurate
	Sdirk2b	2	2	L-stable, stiffly accurate
	Sdirk3a	3	2	L-stable, stiffly accurate
	Sdirk4a [14]	5	4	L-stable, stiffly accurate
	Sdirk4b [14]	5	4	L-stable, stiffly accurate
ROS	Ros2 [26]	2	2	L-stable
	Ros3 [23]	3	3	L-stable
	Rodas3 [23]	4	3	L-stable, stiffly accurate
	Ros4 [14]	4	4	L-stable
	Rodas4 [14]	6	4	L-stable, stiffly accurate

$g(t_F) = \|y - y_{\text{target}}(t_F)\|$. The integral terms can measure the discrepancy between quantities $\mathcal{O}(y)$ predicted by the model and measurements of the same quantities in the physical world, $r = \|\mathcal{O}(y(t)) - z_{\text{observed}}(t)\|$. In the solution of inverse problems the entries of Ψ are referred to as *objective functions*, and in the context of uncertainty quantification as *quantities of interest*.

The output functions (2.6) can be formulated solely in terms of the final state by extending the ODE system. To account for the evolution of the parameters we add the formal equations for the parameter evolution $p' = 0$. To compute the integral terms in (2.6) we add the quadrature variables $q \in \mathbb{R}^o$ whose evolution is defined by $q(t_0) = 0$ and $q' = r\,(t, y, p)$. The equation (1.1) and the outputs (2.6) become

$$\begin{bmatrix} y \\ p \\ q \end{bmatrix}' = \begin{bmatrix} f\,(t,y,p) \\ 0 \\ r\,(t,y,p) \end{bmatrix}, \quad t_0 \le t \le t_F\,; \quad \begin{bmatrix} y(t_0) \\ p(t_0) \\ q(t_0) \end{bmatrix} = \begin{bmatrix} y_0 \\ p \\ 0 \end{bmatrix}, \tag{2.7}$$

$$\Psi = g\,(y(t_F), p) + q(t_F)\,. \tag{2.8}$$

Since (2.6) is equivalent to (2.8), in the remainder of this section we will consider $r = 0$ in order to simplify the presentation, and without loss of generality. However, as explained later, the FATODE implementation treats the case $r \neq 0$ separately to enhance computational efficiency.

Sensitivity analysis yields the derivatives of the model outputs $\Psi_1, \ldots, \Psi_o$, with respect to the model inputs, i.e., the parameters $p_1, \ldots, p_m$. The two main approaches to compute the sensitivity matrix $d\Psi/dp \in \mathbb{R}^{o\times m}$ are discussed next.

2.3. Direct sensitivity analysis. Small changes δp in the parameters result in small perturbations $\delta y(t)$ of the solution of ODE system (1.1), which in turn lead to small changes $\delta\Psi$ in the model outputs. Let $\dot{p} = \delta p / \|\delta p\|$ be the scaled perturbation and $\dot{y} = \delta y / \|\delta p\|$ be the directions of solution change. These directions propagate

forward in time according to the *tangent linear ODE*:

$$\dot{y}' = \boldsymbol{f_y}(t,y;p)\cdot\dot{y} + \boldsymbol{f_p}(t,y;p)\cdot\dot{p}\,,\quad \dot{y}(t_0) = \frac{dy_0}{dp}\cdot\dot{p}\,,\quad \dot{y}(t)\in\mathbb{R}^d\,. \tag{2.9}$$

The sensitivity matrix $d\Psi/dp$ is computed column by column, as follows. Solve the tangent linear model (2.9) with $\dot{p} = e_\ell$ to obtain $\dot{y}_\ell = S_\ell$ (1.2). Here $e_\ell \in \mathbb{R}^m$ is a vector with the ℓth entry equal to one, and all other entries equal to zero. The ℓth column of the sensitivity matrix is

$$\frac{d\Psi}{dp_\ell} = \mathbf{g_y}\left(y(t_F),p\right)\cdot\dot{y}_\ell(t_F) + \mathbf{g_{p_\ell}}\left(y(t_F),p\right)\,,\quad \ell = 1,\ldots,m \tag{2.10}$$

(recall that we now assume $r = 0$ without loss of generality). The computational cost of the forward sensitivity analysis is dominated by the m integrations of the tangent linear model (2.9). Therefore, the forward approach works best when the number of parameters m is small. On the other hand, the number of outputs defines the dimension of (2.10) but has a small influence on (2.9); therefore, it has only a small impact on the total computational cost.

In principle two approaches are possible to obtain the sensitivities in an application. In the *continuous forward sensitivity* approach one first differentiates the forward ODE system (1.1) to obtain the continuous tangent linear ODE (2.9). The forward and the tangent linear ODE systems are solved *together* forward in time. For example, an application of the implicit Euler method leads to the numerical solution

$$y_{n+1} = y_n + h\,f(t_{n+1},y_{n+1},p)\,, \tag{2.11a}$$

$$\dot{y}_{n+1} = \dot{y}_n + h\,\boldsymbol{f_y}(t_{n+1},y_{n+1},p)\cdot\dot{y}_{n+1} + h\,\boldsymbol{f_p}(t_{n+1},y_{n+1},p)\cdot\dot{p}\,. \tag{2.11b}$$

In this case $\dot{y}_n$ represents a numerical approximation of the continuous forward sensitivities $\dot{y}(t_n)$.

In the *discrete forward sensitivity* approach one starts with the numerical approximation of the forward system (2.1) and differentiates it in the direction $\dot{p}$:

$$\dot{y}_{n+1} = \mathbf{\Phi^n_y}(y_n,p)\cdot\dot{y}_n + \mathbf{\Phi^n_p}(y_n,p)\cdot\dot{p},\quad n = 0,\ldots,N-1\,. \tag{2.12}$$

In this case $\dot{y}_n$ represents the sensitivity of the forward numerical solution $dy_n/dp\cdot\dot{p}$.

It is easy to see that the differentiation of the implicit Euler solution (2.11a) with respect to parameters leads to discrete sensitivity equations (2.12) that are precisely (2.11b). Like with implicit Euler, for all methods implemented in FATODE the continuous and the discrete forward sensitivity approaches lead to exactly the same results. FATODE solves the combined equations (2.11) in an efficient manner; for example, in case of implicit schemes, the LU factorization used to solve the sensitivity equation (2.11b) is reused in the next step to solve the forward system (2.11a).

2.4. Adjoint sensitivity analysis. In adjoint sensitivity analysis the matrix $d\Psi/dp$ is computed row by row, as follows. For each output function Ψ_i solve the following *adjoint ODE* for $\lambda_i(t)\in\mathbb{R}^d$ and $\mu_i(t)\in\mathbb{R}^m$:

$$\begin{aligned} \lambda_i' &= -\boldsymbol{f_y}^T(t,y;p)\cdot\lambda_i\,,\quad \lambda_i(t_F) = \boldsymbol{(g_i)_y}^T\left(y(t_F),p\right)\,,\\ \mu_i' &= -\boldsymbol{f_p}^T(t,y;p)\cdot\lambda_i\,,\quad \mu_i(t_F) = \boldsymbol{(g_i)_p}^T\left(y(t_F),p\right)\,;\quad t_F\ge t\ge t_0\,. \end{aligned} \tag{2.13}$$

The adjoint equation (2.13) is derived using variational calculus [19, 20]. We observe that the adjoint equation is solved backward in time and that its formulation depends

on the forward model state $y(t)$. Therefore, in order to solve (2.13) one needs to first solve (1.1) forward in time and save the entire solution $y(t)$. This solution is used to form the Jacobians during the backward in time adjoint integration.

It can be shown that the sensitivity of the ith output with respect to all parameters can be obtained from the adjoint variables as [19, 20]

$$\frac{d\Psi_i}{dp} = \mu_i^T(t_0) + \lambda_i^T(t_0) \cdot \left(\frac{dy_0}{dp} \right) , \quad i = 1, \ldots, o\,.$$

The cost of computing the entire sensitivity matrix $d\Psi/dp$ is dominated by the repeated solution of the adjoint systems (2.13) for $i = 1, \ldots, o$. Therefore, the adjoint method is most effective when the number of outputs o is small. The number m of model inputs (parameters) defines the size of μ_i. Since these variables are obtained via relatively inexpensive Jacobian-transposed vector products ${\mathbf{f_p}}^T \lambda_i$, the number of parameters m does not impact the overall computational cost considerably.

Two approaches are possible to numerically evaluate the sensitivities $d\Psi/dp$. The *continuous adjoint approach* applies a numerical discretization to solve the adjoint ODE (2.13). Since the adjoint ODE is formed first, this strategy is also called the differentiate-then-discretize approach. For example, if the implicit Euler method (2.11) is used to solve (2.13), one obtains

$$\begin{aligned} (\overline{\lambda}_i)_N &= (\boldsymbol{g_i})_{\boldsymbol{y}}^T \left(y(t_F), p\right) ; \quad (\overline{\lambda}_i)_n = (\overline{\lambda}_i)_{n+1} + h\, \boldsymbol{f_y}^T(t_n, y(t_n); p) \cdot (\overline{\lambda_i})_n \,, \\ (\overline{\mu}_i)_N &= (\boldsymbol{g_i})_{\boldsymbol{p}}^T \left(y(t_F), p\right) ; \quad (\overline{\mu}_i)_n = (\overline{\mu}_i)_{n+1} + h\, \boldsymbol{f_p}^T(t_n, y(t_n); p) \cdot (\overline{\lambda}_i)_n \end{aligned} \tag{2.14}$$

for $N - 1 \geq n \geq 0$. In practice the Jacobian arguments are replaced by numerical approximations of the forward trajectory. In general the forward steps and the continuous adjoint steps need not coincide. The Jacobian arguments are computed by interpolating the stored forward solution so as to obtain intermediate state variables at the times required by the backward integration. The continuous adjoint variables are approximations of the solutions of the adjoint ODE (2.13), $(\overline{\lambda}_i)_n \approx \lambda_i(t_n)$, and $(\overline{\mu}_i)_n \approx \mu_i(t_n)$.

In the *discrete adjoint sensitivity* approach one starts with the numerical approximation of the forward system (2.1) and builds the adjoint (linearized transpose) of the discrete system

$$\begin{aligned} \lambda_N &= \boldsymbol{g_y}^T(y_N)\,; & \mu_N &= \boldsymbol{g_p}^T(y_N)\,; \\ \lambda_n &= \boldsymbol{\Phi_y}^n(y_n, p)^T \cdot \lambda_{n+1}; & \mu_n &= \boldsymbol{\Phi_p}^n(y_n, p)^T \cdot \lambda_{n+1}, \quad n = N-1, \ldots, 0\,. \end{aligned} \tag{2.15}$$

For example, the discrete adjoint of the implicit Euler method (2.11) reads

$$\begin{aligned} (\lambda_i)_N &= (\boldsymbol{g_i})_{\boldsymbol{y}}^T \left(y_N, p\right) ; \quad (\lambda_i)_n = (\lambda_i)_{n+1} + h\, \boldsymbol{f_y}^T(t_{n+1}, y_{n+1}; p) \cdot (\lambda_i)_n \,, \\ (\mu_i)_N &= (\boldsymbol{g_i})_{\boldsymbol{p}}^T \left(y_N, p\right) ; \quad (\mu_i)_n = (\mu_i)_{n+1} + h\, \boldsymbol{f_p}^T(t_{n+1}, y_{n+1}; p) \cdot (\lambda_i)_n \end{aligned} \tag{2.16}$$

for $N - 1 \geq n \geq 0$. The arguments of $\boldsymbol{f_y}^T$, $\boldsymbol{f_p}^T$ are the numerical approximations of the implicit Euler method applied to the forward problem (2.11). The discrete adjoint approach (2.16) follows exactly the same sequence of time steps as the forward integration (2.11), but in reverse order. The discrete adjoint variables represent derivatives of the numerical solution, e.g., $(\lambda_i)_n = d\Psi_i(y_N)/dy_n$.

The continuous (differentiate-then-discretize) and the discrete (discretize-then-differentiate) adjoint approaches lead, in general, to different computational results

[25]. This can be easily seen for our implicit Euler example by comparing (2.14) and (2.16). The arguments of $\boldsymbol{f}_{\boldsymbol{y}}^T$, $\boldsymbol{f}_{\boldsymbol{p}}^T$ differ; even if the same sequence of forward steps is used in both cases, the Jacobians are evaluated at y_n in the continuous adjoint and at y_{n+1} in the discrete adjoint systems.

All sensitivity packages currently available (Odessa [17], CVODES [24], and DENSERKS [2]) implement a continuous adjoint sensitivity approach. Important characteristics of the continuous adjoint approach are as follows. Continuous adjoints are numerical solutions of the adjoint ODE; therefore, they always approximate the continuous derivatives. However, they are not the (exact) gradients of any function. The continuous approach offers implementation flexibility; the numerical method and the sequence of step sizes used to solve the adjoint ODE may differ from those used to solve the forward ODE and may be tuned separately to satisfy the accuracy needs of the reverse integration. However, this comes at a cost; the forward numerical solution needs to be interpolated to the time points required in the adjoint integration.

The discrete adjoint approach has several important characteristics that distinguish it from the continuous approach. Discrete adjoint variables are derivatives of the forward numerical solution. However, there is no guarantee that they are approximations of the continuous derivatives [19, 21]. The discrete approach provides exact gradients (within roundoff) of the numerical cost functions, which is advantageous in the solution of numerical optimization problems. The computational process and the sequence of step sizes are determined by the forward numerical method and by the forward steps; there is no flexibility to separately tune the adjoint integration. In the same time, all the variables needed to form the discrete adjoint are computed during the forward solution, and no additional interpolations are necessary. The human effort required to implement the adjoint of a complex model is considerable; discrete adjoints can be constructed automatically by algorithmic differentiation [4] to reduce this effort. In addition, one can compute sensitivities of ODEs described by legacy code, and for which the analytical formulation may be unavailable.

FATODE is the first package to implement discrete adjoints for all the methods considered. This allows the users to benefit from all the advantages of this approach described above. In addition, the discrete adjoints implemented have good theoretical properties in the sense that they are consistent discretizations of the adjoint ODE. We recall the following theorem from [19, 20].

THEOREM 2.1 (consistency of discrete Runge–Kutta adjoints). *If a Runge–Kutta method* (2.2) *has order of consistency p, then its discrete adjoint* (2.15) *is an order p discretization of the adjoint ODE* (2.13).

The proof of this theorem accounts for both the adjoint truncation error and the approximation of the linearization point (the use of a forward discrete solution y_n instead of $y(t_n)$ in the Jacobian arguments). Similar arguments apply to the discrete adjoints of Rosenbrock methods (2.4).

3. Selection of different options in FATODE. FATODE offers a wide array of tunable parameters for the solution of ODEs and the calculation of sensitivities. We provide some general guidelines that could help a user select the options that best suit the problem at hand. A complete discussion of the available options is given in the FATODE *User's Guide* [29].

3.1. Selecting a family of methods. ERK are the methods of choice for nonstiff problems. SDIRK and Rosenbrock are preferable for stiff problems where a moderate accuracy is required, and FIRK methods are for very stiff systems or when a high accuracy is needed. Among the implicit schemes, the FIRK cost per step is

the highest, while the Rosenbrock cost per step is the lowest.

3.2. Selecting a particular method within a family. This choice reflects the tradeoff between compute time and accuracy. For maximum efficiency one would choose schemes of order two to four for faster, low to medium accuracy calculations, and progressively higher order schemes for more accurate results. Methods of order five to eight are most efficient when high accuracies are sought.

As explained in section 2.1, all L-stable methods are suitable for the integration of stiff systems as they provide strong damping of the fast components of the error. Methods which, in addition, are stiffly accurate retain the solution accuracy in the limit of infinite stiffness. Therefore, stiffly accurate methods like Radau2A, Lobatto3C, Sdirk4{a,b}, and Rodas{3,4} are well suited for the integration of very stiff systems (1.1). On the other hand, the Gauss method is only weakly A-stable, meaning that it only weakly attenuates fast transients, and is therefore not suited for very stiff systems. The Gauss method is useful when the energy of the system needs to be preserved, for example, in the case of Hamiltonian dynamics.

3.3. Selecting a linear algebra solver. The most computationally intensive part in solving large-scale ODE systems by implicit methods is the solution of linear systems at each step. FATODE provides several direct linear algebra solvers that work well for small and medium-size problems. The selection of a specific package depends on the problem at hand: LAPACK should be used for problems with full Jacobian matrices, and UMFPACK or SuperLU should be used for problems with a sparse structure. For very large problems a better solution is to link the integrators to third-party Krylov space iterative solvers and utilize problem-specific preconditioners.

3.4. Choosing the direct or the adjoint approach to sensitivity calculations. The two main approaches to compute the sensitivity matrix $d\Psi/dp \in \mathbb{R}^{o\times m}$ have different computational complexities, as explained in section 2. As a rule of thumb, the direct (or tangent linear) method is chosen when the number of parameters is smaller than the number of outputs ($m \ll o$), while the adjoint method is chosen when the number of parameters is larger than the number of outputs ($m \gg o$). When $m \approx o$ the direct method is preferable due to its lower implementation complexity.

For the adjoint method the user has the option to compute sensitivities with respect to only the initial conditions, e.g., for a data assimilation application, or to compute sensitivities with respect to both model parameters and initial conditions, e.g., in a parameter estimation application.

3.5. Selecting the solution approach for the sensitivity systems. Both the tangent and the adjoint ODEs are linear, and during each step the corresponding sensitivity variables are the solutions of a linear system of equations. FATODE offers the choice to build this system and to solve it directly or to employ simplified Newton iterations that reuse the LU decompositions performed during forward calculation. If the LU factors can be stored, then the simplified Newton iterations save considerable CPU time in forward sensitivity mode by reusing the LU decomposition. In adjoint sensitivity calculations the computational savings need to be judged against the additional read/write overhead for checkpointing large, multiple LU factorized matrices, and against the physical dimensions of the tape. The direct approach should be selected if an iterative linear algebra solver is employed, and no explicit LU decomposition is available. When simplified Newton iterations are used the computed adjoints are equal to the discrete gradients within the truncation error margin (and not to roundoff error).

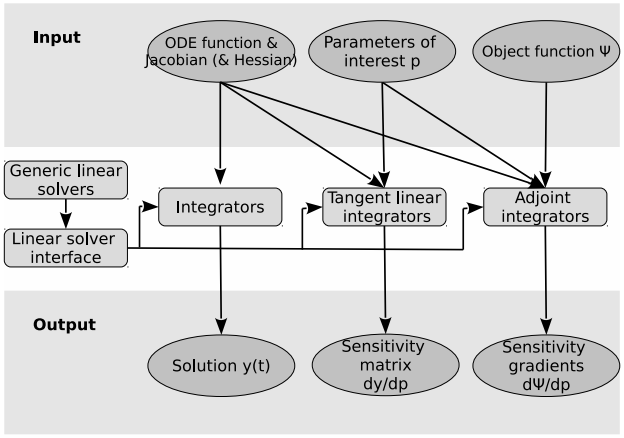


FIG. 1. *Overall structure of* FATODE.

3.6. Choosing tolerances. Absolute and relative tolerances reflect the desired degree of solution accuracy. It is a good idea to select slightly tighter tolerances for sensitivities than required by the forward application alone. For best performance absolute tolerances need to be chosen such as to reflect the magnitude of each solution component. This is typically very difficult with sensitivity calculations, as sensitivity coefficients can vary over many orders of magnitude, and little information about their values is available a priori. Repeated calculations may be needed to properly calibrate absolute tolerances.

3.7. Computing derivatives. The stiff solvers, as well as the tangent linear and adjoint methods, require the computation of various derivatives such as the Jacobians of the ODE function or of the output functions with respect to the state and parameters. The derivatives supplied to FATODE can be obtained analytically, by finite differences, or by automatic differentiation. Details about the required derivatives and their implementation can be found in [31, part G.1]. Analytical or automatic differentiation generated Jacobians and Hessians are to be preferred due to their accuracy and, most often, their computational efficiency. Finite differences are the least recommended as their accuracy is difficult to control, and their errors directly impact the computed sensitivities. Rosenbrock methods require the most additional derivatives, including Hessians, for sensitivity calculations. If derivatives are difficult to obtain, or expensive to run, then SDIRK or FIRK methods are to be preferred.

4. Code organization. FATODE implements four types of methods: explicit, fully implicit, and singly diagonally implicit Runge–Kutta methods, and Rosenbrock methods. For each family of methods, a module is given for the main integrator, a module is given for the linear system solver interface, and a set of modules is given for generic linear system solvers. They form the basic structure shown in Figure 1.

The main integration module provides the basic time stepping framework and is independent of the linear system solver.

- The forward integrator calls the user-supplied right-hand side function and Jacobian and accesses the linear system solver in order to compute the ODE solution. Users can employ FATODE as a high quality ODE solution library even when its sensitivity analysis capabilities are not needed. Details of the

implementation of different Runge–Kutta and Rosenbrock methods are given in [31, part B].

- The tangent linear integrator and the adjoint integrator require users to also specify the parameters of interest as additional inputs. The tangent linear integrator, by default, considers the initial conditions of the ODE system as the parameters of interest and computes the sensitivity of the ODE solution with respect to them. Highly efficient tangent linear implementations are obtained by reusing the LU decompositions from the forward solution on the sensitivity equations [8]. Details about each family of tangent linear methods are given in [31, part C].
- For efficiency reasons FATODE provides two implementations of the adjoint integrator—one for sensitivities with respect to initial conditions, detailed in [31, part D], and one for sensitivities with respect to parameters, discussed in [31, part F]. By default scalar cost functions (2.6) are considered. The cost function and its derivatives are supplied by the user; cost function derivatives are used to define the adjoint initial conditions (see (2.15) and [31, eqn. (E.6)]) and to provide the forcing of the adjoint equations.

The linear system solver modules define the data structures for the Jacobians (e.g., dense, sparse), interfaces to routines that compute Jacobian (or its transpose) times vector products, and interfaces to linear solvers. Linear algebra is implemented transparently to the ODE solvers via four generic routines: *LS_Init* (allocates memory and initializes the specific linear solver), *LS_Decomp* (LU decomposition), *LS_Solve* (solves the triangular systems by substitution), and *LS_Free* (frees the memory allocated during the initialization stage). They provide interfaces to dense (LAPACK [3]) and sparse (UMFPACK [6] and SUPERLU [7]) linear algebra packages. Separate modules are provided for each of the implicit time stepping families in FATODE, as they perform different Jacobian operations and solve different types of systems (e.g., real-valued linear systems of dimension $d \times d$ for Rosenbrock and real and complex linear systems of dimension $d \times d$ for Runge–Kutta). All the required code modifications for a new application, or for adding a new solver, are done within the linear algebra modules. Details on the linear algebra part of FATODE are given in [31, part H].

The adjoint model requires two runs. First, the forward code performs a regular integration of the ODE system. At each step the time t_n, the step size, the forward solution vector, and the intermediate stage vectors are all saved in checkpoints. Next, the discrete adjoint code is run backward in time and traces the same sequence of steps, but in reverse order. At each step, the data in the checkpoint storage is retrieved and used to construct the adjoint system. The results of LU factorizations can, in principle, be checkpointed and reused in the adjoint calculations. This is not done in our implementation for the following reasons. The memory and input/output costs for storing LU factors can be extremely large when the system is large or when many steps are taken. Next, FATODE is designed such that the details of the linear solver are transparent to the main integrator. This transparency cannot be preserved when one builds and manages a stack for data structures that are specific to particular linear solvers.

Variable step size is employed by FATODE to control numerical errors and maximize efficiency. The forward integrators estimate the truncation errors using Runge–Kutta and Rosenbrock embedded solutions. Local error tests are performed based on the relative and absolute error tolerances specified by the user, and they are used to decide acceptance/rejection of the current solution as well as the size of the next step. For the forward sensitivity analysis FATODE allows control of the truncation

errors for both the forward solution and the tangent linear model solution. There is no step size control during the discrete adjoint integration as it traces the same sequence of steps as the forward integration. However, user-specified tolerances are employed to control the convergence of the iterative solutions of sensitivity equations. Implementation details for error estimates and step size control are given in [31, part G].

To increase efficiency the FIRK and SDIRK implementations have the possibility to reuse previous LU factorizations in lieu of recomputing a new factorization at the current step. The reuse is allowed when the following three conditions are simultaneously met: the same LU factorization has not been in use for more than $N_{\max}$ consecutive steps; the ratio of the predicted step size to the current step size satisfies $q_{\min} \leq h_{\text{predicted}}/h \leq q_{\max}$; and the convergence rate of the simplified Newton iterations in the previous step is smaller than $\theta_{\max}$. The default values are $N_{\max} = 20$, $q_{\min} = 1$, $q_{\max} = 1.2$, and $\theta_{\max} = 0.001$, and they can be changed by the user.

5. Numerical experiments with a nonstiff problem: Two-dimensional shallow water equations. In this section we illustrate the capabilities of FATODE and evaluate the performance of each family of methods using a semidiscretized system of partial differential equations. We compare FATODE with the well-established code CVODES within SUNDIALS [15] for the forward solution, as well as for the direct and adjoint sensitivity analysis. Note that CVODES is implemented in the C programming language, while FATODE is written in Fortran. To reduce the influence of different compilers, we use the same Fortran implementations of the right-hand side function and its Jacobian and call them from CVODES via C interfaces.

All the experiments are performed on a workstation with dual Intel Xeon E5-2630 CPUs (2.3GHz) running Fedora 17 (x86_64) Linux. PGI Fortran Version 12.10-5 and GCC Version 4.7.2 are used for compilation, with level O3 optimization. Unless stated otherwise, the default settings for parameters are used in all FATODE calls.

We consider the two-dimensional shallow water equations [16]

$$
\begin{aligned}
&\frac{\partial}{\partial t} h + \frac{\partial}{\partial x}(uh) + \frac{\partial}{\partial y}(vh) = 0,\\
&\frac{\partial}{\partial t}(uh) + \frac{\partial}{\partial x}\left(u^2 + \frac{1}{2}gh^2\right) + \frac{\partial}{\partial y}(uvh) = 0,\\
&\frac{\partial}{\partial t}(vh) + \frac{\partial}{\partial t}(uvh) + \frac{\partial}{\partial y}\left(v^2 h + \frac{1}{2}gh^2\right) = 0,
\end{aligned}
\tag{5.1}
$$

where $u(t,x,y)$, $v(t,x,y)$ are the components of the velocity field, $h(t,x,y)$ is the fluid layer thickness, and g denotes the gravitational acceleration. The spatial domain is $\Omega = [-3,\, 3]^2$, and the simulation time interval is $[t_0, t_F] = [0, 0.02]$ time units. The spatial domain is covered by a grid of size 40×40, and third order upwind finite differences are used to perform the spatial discretization. This results in a large, nonstiff ODE system of dimension $40 \times 40 \times 3 = 4800$ which is solved by FATODE.

A reference solution of the ODE system is obtained by using LSODE [18], a well-known but relatively slow ODE solver, with a very tight relative and absolute tolerances of 10^{-14}. The solution relative error is defined as

$$
Err = \|y(t_F) - y_{\text{ref}}(t_F)\|_2 \,/\, \|y_{\text{ref}}(t_F)\|_2 \ , \tag{5.2}
$$

where $y(t_F)$ is the numerical solution at the final step t_F, and $y_{\text{ref}}(t_F)$ is the reference

solution at t_F. A similar metric is used for the relative sensitivity errors

$$Err = \|s - s_{\text{ref}}\|_2 \,/\, \|s_{\text{ref}}\|_2 \,, \tag{5.3}$$

where s is a numerical sensitivity vector. The reference values s_{ref} are computed using Adam–Moulton methods in CVODES [15] with relative and absolute tolerances of 10^{-14} for adjoint sensitivities, and with a relative tolerance of 10^{-9} and an absolute tolerance of 10^{-10} for tangent linear sensitivities.

We report below numerical results with the nonstiff integrators in FATODE and CVODES. Results with the stiff solvers in FATODE and CVODES on the shallow water test can be found in the supplementary material [31, part I].

5.1. Forward solution. CVODES [15] is the sensitivity analysis-enabled version of CVODE, which uses the Adams–Moulton methods for nonstiff ODE systems and the backward differentiation formulas (BDFs) for stiff ODE systems. Both methods are implemented in a variable-order variable-step form. In our test, we employ the Adams–Moulton method in CVODE. We select three ERK methods of orders three, five, and eight in FATODE. The Gustafsson predictive error controller is used for all of them.

With each solver we vary both absolute and relative error tolerances from 10^{-2} to 10^{-7} to obtain solutions of different levels of accuracy (the absolute and relative error tolerances are equal to each other). The resulting work-precision diagrams are shown in Figures 2(a) (error versus step size) and 2(b) (error versus CPU time). For the same level of accuracy, the ERK methods in FATODE take fewer steps than the Adams–Moulton method in CVODES, but the cost per step is higher. Nevertheless, FATODE's ERK methods of orders five and eight outperform CVODES, with the best overall CPU time being achieved by FATODE's order eight method.

5.2. Direct sensitivity analysis. We now calculate the sensitivities of all solution components at the final time with respect to the initial value of the first solution component $\partial y_i(t_F)/\partial y_1(t_0)$, $i = 1, \ldots, d$, using tangent linear model integration.

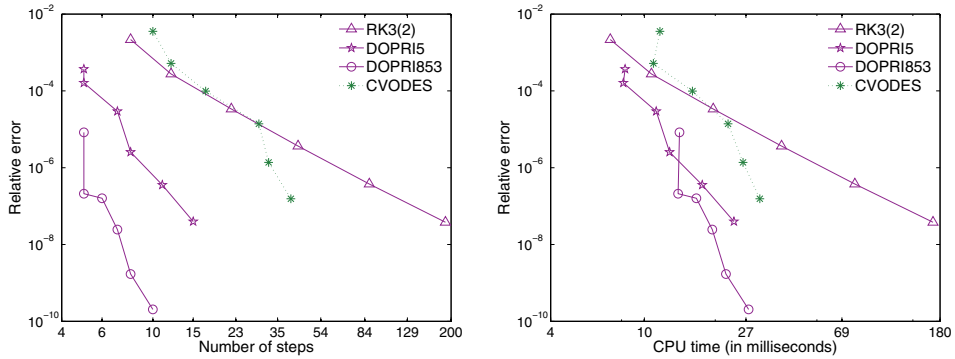
The performance results of nonstiff tangent linear integrators are shown in Figures 2(c) (error versus step size) and 2(d) (error versus CPU time). FATODE's ERK TLM methods of all orders are considerably more efficient than the nonstiff forward sensitivity solver in CVODES in terms of both CPU time and number of steps.

5.3. Adjoint sensitivity analysis. We next calculate the sensitivities of the first solution component at the final time with respect to all initial values $\partial y_1(t_F)/\partial y_i(t_0)$, $i = 1, \ldots, d$, using adjoint model integration.

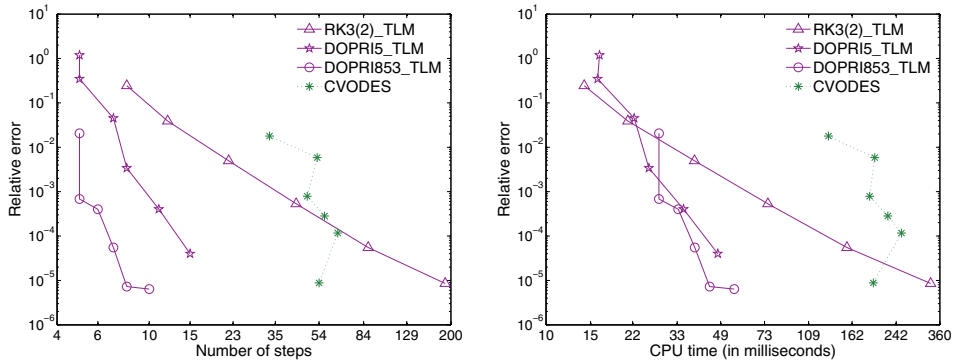
Work-precision diagrams for the adjoint explicit solvers are shown in Figures 2(e) (error versus step size) and 2(f) (error versus CPU time). The ERK adjoint methods in FATODE take fewer steps but run slightly more slowly than the Adams–Moulton method in CVODES. This is due to the fact that the ERK method in FATODE requires more Jacobian evaluations and Jacobian-vector products during the adjoint backward run.

6. Numerical experiments with a very stiff problem: The carbon bond chemical system. We now apply FATODE to the integration and parameter sensitivity analysis for a very stiff system. The experiments are carried out on the same platform and with the same Fortran compiler and optimization options as for the shallow water test. The relative errors are evaluated using formulas (5.2) and (5.3).

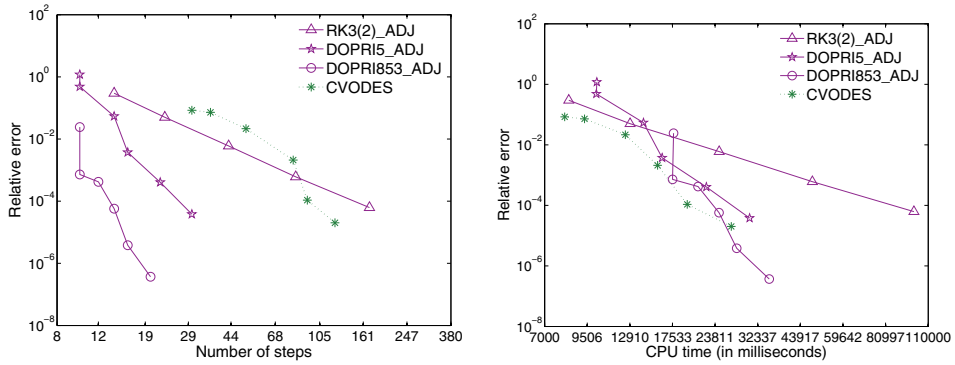
The Carbon Bond-IV Mechanism (CBM-IV), which consists of 32 species and 81 reactions, was developed for simulating urban smog and modeling regional atmo-



(a) ODE solution error vs. number of steps (b) ODE solution error vs. CPU time



(c) Forward sensitivity error vs. number of steps (d) Forward sensitivity error vs. CPU time



(e) Adjoint sensitivity error vs. number of steps (f) Adjoint sensitivity error vs. CPU time

FIG. 2. *ODE solution and sensitivity analysis of the shallow water equations* (5.2) *using nonstiff solvers. Comparison is made between three ERK methods in* FATODE *and the Adams–Moulton method in* CVODE. *Different points in each plot correspond to different tolerance levels in the range* $10^{-2}, 10^{-3}, \ldots, 10^{-7}$ *(with the absolute tolerances equal to the relative tolerances). The tangent linear models compute the sensitivity* $\partial y_i(t_F)/\partial y_1(t_0),\ i = 1, \ldots, d$, *and the adjoint models compute the sensitivity* $\partial y_1(t_F)/\partial y_i(t_0),\ i = 1, \ldots, d$.

spheric pollution [5]. Complete descriptions of the CBM-IV chemical mechanism, and of the setting of the numerical experiments, are given in the supplementary material [31, part J]. The first six dominant eigenvalues of the Jacobian matrix vary in magni-

tude from -1.4×10^9 to -2.4×10^{-1}, which indicates that the system is very stiff and requires implicit methods with good stability properties. We select two L-stable and stiffly accurate methods (a high order one and a low order one) from each of the three categories of implicit methods in FATODE. Their performance is compared against the BDF method implemented in CVODES.

The sensitivities of individual species concentrations at the final time with respect to a variety of reaction rate coefficients are calculated using adjoint integration. These sensitivities quantify the change in concentrations due to small perturbations in the reaction rate coefficients and help identify the most important reactions. Of particular interest is the response of the gas-phase species O_3, NO_2, $HONO$, N_2O_5, and HNO_3 to perturbations of reaction rates.

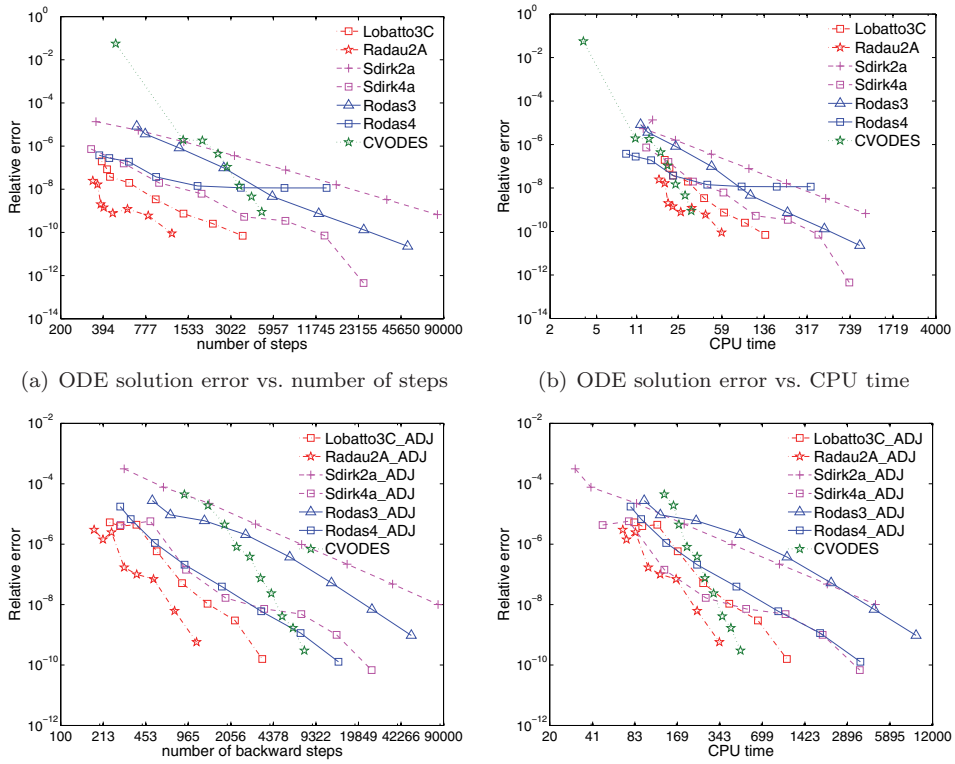
To obtain a reference solution, both adjoint CVODES and the adjoint RADAU5 integrator in FATODE are run with very tight tolerance settings. For the ODE solution we use $rtol = 10^{-14}$ and $atol = 10^{-8} \times atol_0$ for both codes. The entries of the vector of absolute tolerances $atol_0$ reflect the magnitude of typical concentrations of individual chemical species. CVODES requires additional tolerances for sensitivity analysis, which are set to $rtol\_sen = 10^{-14}$, $atol\_sen = 10^{-17}$. The two codes yield sensitivity results that are very close to each other. To provide the most favorable comparison setting for CVODES we use its results as the reference solutions in (5.2) and (5.3).

For the numerical tests each solver is run with a series of progressively tighter tolerances $\{atol, rtol\} = \{atol_0/(5^k), 10^{-3}/(5^k)\}$ for $k = 0, \ldots, 7$ (for CVODES we use $k = 0, \ldots, 9$ to better capture its behavior at high accuracy levels). The forward ODE integration results shown in Figures 3(a), 3(b) reveal that the most efficient solvers are FATODE's Rodas4 (for lower accuracies) and Radau2A (for higher accuracies), followed closely by CVODES.

The work-precision diagrams for the adjoint sensitivity results are shown in Figure 3(c) (with respect to the number of backward steps) and in Figure 3(d) (with respect to CPU time). For FATODE the number of backward steps is the same as the number of accepted steps during forward integration, while for CVODES the number of backward steps varies for each different adjoint problem. Figure 3(c) reveals that the number of backward steps in FATODE is in general smaller that in CVODES. The results in Figure 3(d) show that for moderate accuracies (relative error above 10^{-5}) the SDIRK and the Rosenbrock adjoint methods in FATODE are more effective than CVODES. For high accuracies the most efficient is FATODE's FIRK adjoint method Radau2A. Note that CVODES implements a variable-order BDF method with maximum order five. The most efficient method Radau2A has order five as well, while the other schemes in FATODE have lower orders.

7. Comparison of different option choices in FATODE. FATODE offers a multitude of methods and a wide range of tunable parameters for each method. This allows the user to configure the solvers to best meet the needs of the application at hand. This section discusses several important choices of methods and parameters in FATODE and illustrates their impact on the solver performance with the help of the shallow water and carbon bond example problems. For a detailed discussion of all available choices the reader should consult FATODE's *User's Guide* [29].

7.1. Choice of method. Generally speaking, higher order methods are more efficient than lower order methods when medium to high accuracies are sought. Figure 2 illustrates this in the context of the shallow water problem: the eighth-order DOPRI853 is the most efficient explicit method for both the ODE solution and the sensitivity analysis.



(a) ODE solution error vs. number of steps

(b) ODE solution error vs. CPU time

(c) Adjoint sensitivity error vs. number of backward steps

(d) Adjoint sensitivity error vs. CPU time

FIG. 3. *Work-precision diagrams for the ODE solution and for adjoint sensitivities for the stiff CBM-IV test problem. Adjoint sensitivities of five chosen species (O_3, NO_2, $HONO$, N_2O_5, and HNO_3) with respect to* 24 *constant reaction rate coefficients are computed.*

In case of implicit schemes the solution of nonlinear systems dominates the total cost when large ODE systems are solved. The test results on the shallow water equations (reported in the supplementary material) reveal that Sdirk4a is the most efficient implicit solver for both forward and tangent linear integration. For smaller ODE systems like CBM-IV, the highest efficiency can be obtained by FIRK methods, e.g., Radau2A in Figure 3. The sixth-order Gauss method takes too many steps to be competitive on CBM-IV. This is due to its weak A-stability. For larger systems the stage coupling inherent with FIRK methods impacts the cost of the linear solvers, which explains why SDIRK and Rosenbrock methods perform best on the shallow water test.

Results with both examples confirm the fact that Rosenbrock methods are particularly favorable for low accuracy requirements. Ros4 and Ros3 have the same orders with Rodas4 and Rodas3, but take fewer stages, and thus are expected to behave similarly on problems that are mildly stiff. Additional tests with CBM-IV confirm that on very stiff problems the stiffly accurate schemes Rodas4 and Rodas3 perform considerably better than Ros4 and Ros3, respectively.

7.2. Choice of linear solver. We have assessed the efficiency of sparse linear solvers incorporated in FATODE by comparing their performance with LAPACK full

linear solvers in the shallow water test. The Jacobian matrix is of dimension 4800, but only 100,800 elements (0.4375%) are nonzero. The incorporation of sparse linear solvers leads to significant computational savings and allows for very efficient forward, tangent linear, and adjoint model integration. The compute times for various implicit integrators in FATODE using different linear solvers are shown in Table 2. As expected, FATODE benefits significantly from the use of sparse linear solvers. UMFPACK is slightly faster than SUPERLU for this test, and both sparse solvers give essentially the same results as the full algebra linear solver. Users can link their own linear algebra routines, e.g., preconditioned Krylov based iterative methods.

TABLE 2

Overall compute times (in seconds) using different linear solvers in FATODE *for the shallow water test problem. The tolerances are atol = rtol = 10^{-6}, and the time interval is $t_0 = 0$ to $t_F = 0.02$ (time units).*

Solver	Full algebra	Sparse algebra	
	LAPACK	UMFPACK	SUPERLU
Ros4	474.7	21.6	31.7
Ros4_TLM	506.8	58.3	71.4
Ros4_ADJ	991.7	96.8	120.2
Lobatto3C	342.0	8.8	14.8
Lobatto3C_TLM	431.3	32.9	40.0
Lobatto3C_ADJ	1722.4	69.0	102.2
Sdirk4a	74.5	3.8	5.7
Sdirk4a_TLM	125.2	46.0	49.0
Sdirk4a_ADJ	1685.9	69.0	107.4

7.3. Choice of the direct versus adjoint approach. In the shallow water test, we used the tangent linear model to calculate $\partial y_i(t_F)/\partial y_1(t_0)$, $i = 1, \ldots, d$ (with one tangent linear variable) and the adjoint model to calculate $\partial y_i(t_F)/\partial y_1(t_0)$, $i = 1, \ldots, d$ (with one adjoint variable). These two vectors correspond to the first column and the first row of the full sensitivity matrix $(\partial y_i(t_F)/\partial y_j(t_0)) \in \mathbb{R}^{d\times d}$, respectively. The two approaches compute the same sensitivities: the tangent linear result and the adjoint result converge to the same value for $\partial y_1(t_F)/\partial y_1(t_0)$. The computational costs are, however, different. The adjoint approach is considerably more efficient when only a small number of rows in the sensitivity matrix is needed, and the direct approach is considerably more efficient when only a few columns are computed.

7.4. Solution of the sensitivity system. Both the forward sensitivity system (2.9) and the adjoint sensitivity system (2.13) are linear with respect to the corresponding sensitivity variables. Consequently, at each step of an implicit solver, the stage variables are the solutions of a linear system. FATODE offers two choices to solve these linear systems: directly, at the expense of additional LU decompositions at each step, and iteratively, via simplified Newton iterations that reuse the LU decomposition for each stage.

The relative efficiency of the two choices is both problem dependent and method dependent. The iterative approach is preferable for large problems, while the direct approach works best with smaller systems. For the s-stage FIRK methods the direct solution involves one coupled system of dimension $sd \times sd$, while for SDIRK methods there are s systems of dimension $d \times d$. Considering the cost scaling, the direct approach is the default option for SDIRK methods, and the iterative approach is the

default for FIRK methods. This can be changed by users to best match the problem at hand.

8. Conclusions. This paper presents the FATODE library for the integration of stiff ODE systems and for performing direct and discrete adjoint sensitivity analyses. The software is based on our KPP numerical library; while KPP solves only chemical kinetic systems, FATODE offers a general purpose implementation that makes it potentially useful for a wide range of applications. Important areas that can benefit from using FATODE include uncertainty quantification, inverse problems such as parameter estimation and data assimilation, and optimization of systems governed by ODEs.

FATODE implements ERK methods for nonstiff problems and FIRK, SDIRK, and Rosenbrock methods for stiff problems. This distinguishes the software from CVODES, which is based on linear multistep methods.

FATODE employs a *discrete adjoint sensitivity analysis* approach; i.e., it computes the derivatives of the numerical solution. This approach sets it apart from both DENSERKS and CVODES, which implement a continuous adjoint approach. The discrete adjoint approach gives gradients of the numerical cost function that are exact, to roundoff error. Such gradients are highly suitable for numerical optimization problems, as well as for a posteriori error estimation in complex systems. It was shown in [20] that the discrete Runge–Kutta adjoints are dual consistent. Consequently, the discrete adjoints computed by FATODE represent solutions of the continuous adjoint ODEs that are as accurate as the solutions obtained by the continuous adjoint approach. We note that controlling the sensitivity error may require several runs. Since sensitivity coefficients can vary over many orders of magnitude, a good setting of the absolute tolerances may require several trial-and-error iterations. In the adjoint approach, the sensitivity error depends on both the forward and the adjoint integration errors; this issue is common to both continuous and discrete approaches. While the independent adjoint step size control in the continuous approach is apparently an advantage, current implementations do not account for the interpolation errors. Moreover, in practice, the interplay between forward, interpolation, and adjoint discretization errors is very difficult to control directly. A good guideline is to use tighter tolerances than required by the application during the sensitivity calculations.

All methods in FATODE use local truncation error estimation and step size control for efficiency. The step size is also adjusted to ensure rapid convergence of the linear and nonlinear system solvers at each step. Checkpointing, needed for adjoint runs, is performed in a manner that is completely transparent to the user. The discrete implicit sensitivity equations require the solution of different linear systems for each stage, or of linear systems that couple all stages of a method. Two options are offered in FATODE: the first constructs each linear system and solves it individually, and the second performs simplified Newton iterations that allow reusing the same LU factorization across all stages. FATODE controls the iteration number, and possibly the step size, such that the iteration error in the latter approach is smaller than the local truncation error at the current step.

FATODE contains stand-alone linear system solvers for different types of systems. The package includes direct solvers for both dense (LAPACK) and sparse (SUPERLU, UMFPACK) systems. The decoupling between the time integration routines and the linear solvers allows users to easily add their own methods, e.g., preconditioned iterative solvers tuned for the application at hand. Details can be found in the *User's Guide* [29].

The implicit formulation of the stiff solvers, as well as the formulation of the

tangent linear and adjoint methods, require the user to provide various derivatives such as Jacobian-vector, transposed Jacobian-vector, and Hessian-vector products. These derivatives can be obtained analytically, by finite differences, or by automatic differentiation. Since the accuracy of the derivative approximations directly impacts the overall accuracy of the sensitivity results, care must be exercised when finite differences are employed.

Numerical experiments presented here reveal that FATODE compares favorably with the current state-of-the-art package CVODES. For the (larger) shallow water problem in section 5 the stiff FATODE solvers are considerably more efficient, in terms of work-precision, than the BDFs implemented in CVODES. For nonstiff solvers the linear multistep methods provide better performance. FATODE sensitivity solutions are considerably more accurate than those of CVODES for the stiff chemical system discussed in section 6.

Code availability. The source code and the *User's Guide* are available online from the FATODE web site [30]. The code has been tested under the following compilers: Portland group's pgf90, Lahey's lf95, Sun's sunf90, gfortran, g95, and Absoft.

REFERENCES

[1] A. SANDU, D. DAESCU, AND G. R. CARMICHAEL, *Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: Part* II*—numerical validation and applications*, Atmospheric Environment, 37 (2003), pp. 5097–5114.

[2] M. ALEXE AND A. SANDU, *Forward and adjoint sensitivity analysis with continuous explicit Runge-Kutta schemes*, Appl. Math. Comput., 208 (2009), pp. 328–334.

[3] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. DU CROZ, S. HAMMARLING, A. GREENBAUM, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[4] H. M. BÜCKER, G. CORLISS, P. HOVLAND, U. NAUMANN, AND B. NORRIS, EDS., *Automatic Differentiation: Applications, Theory, and Implementations*, Lect. Notes Comput. Sci. Eng. 50, Springer, New York, 2006.

[5] W. P. L. CARTER, *A detailed mechanism for the gas-phase atmospheric reactions of organic compounds*, Atmospheric Environment, 24 (1990), pp. 481–518.

[6] T. A. DAVIS, *Algorithm* 832: *UMFPACK V*4.3*—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Software, 30 (2004), pp. 196–199.

[7] J. W. DEMMEL, S. C. EISENSTAT, J. R. GILBERT, X. S. LI, AND J. W. H. LIU, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[8] A. M. DUNKER, *The decoupled direct method for calculating sensitivity coefficients in chemical kinetics*, J. Chem. Phys., 81 (1984), pp. 2385–2393.

[9] P. ELLER, K. SINGH, A. SANDU, K. BOWMAN, D. K. HENZE, AND M. LEE, *Implementation and evaluation of an array of chemical solvers in the Global Chemical Transport Model GEOS-Chem*, Geoscientific Model Development, 2 (2009), pp. 89–96.

[10] M. W. GERY, G. Z. WHITTEN, J. P. KILLUS, AND M. C. DODGE, *A photochemical kinetics mechanism for urban and regional scale computer modeling*, J. Geophys. Res., 94 (1989), pp. 12925–12956.

[11] R. GIERING AND T. KAMINSKI, *Recipes for adjoint code construction*, ACM Trans. Math. Software, 24 (1998), pp. 437–474.

[12] W. HAGER, *Runge Kutta methods in optimal control and the transformed adjoint system*, Numer. Math., 87 (2000), pp. 247–282.

[13] E. HAIRER, S. P. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations* I. *Nonstiff Problems*, Springer-Verlag, Berlin, 1993.

[14] E. HAIRER AND G. WANNER, *Solving Ordinary Differential Equations* II. *Stiff and Differential-Algebraic Problems*, Springer Ser. Comput. Math., Springer, Berlin, 1991.

[15] A. C. HINDMARSH, P. N. BROWN, K. E. GRANT, S. L. LEE, R. SERBAN, D. E. SHUMAKER, AND C. S. WOODWARD, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*, ACM Trans. Math. Software, 31 (2005), pp. 363–396.

[16] D. D. HOUGHTON AND A. KASAHARA, *Nonlinear shallow fluid flow over an isolated ridge*, Comm. Pure Appl. Math., 21 (1968), pp. 1–23.

[17] J. R. Leis and M. A. Kramer, *ODESSA—an ordinary differential equation solver with explicit simultaneous sensitivity analysis*, ACM Trans. Math. Software, 14 (1986), pp. 61–75.

[18] K. Radhakrishnan and A. C. Hindmarsh, *Description and Use of lsode, the Livermore Solver for Ordinary Differential Equations*, Tech. report UCRL-ID-113855, Lawrence Livermore National Laboratory, Livermore, CA, 1993.

[19] A. Sandu, *On the properties of Runge Kutta discrete adjoints*, in ICCS 2006, IV, Lecture Notes in Comput. Sci. 3994, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 550–557.

[20] A. Sandu, *Solution of inverse problems using discrete ODE adjoints*, in Large Scale Inverse Problems and Quantification of Uncertainty, John Wiley and Sons, New York, 2011, pp. 345–364.

[21] A. Sandu, D. Daescu, and G. R. Carmichael, *Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: Part* I*—theory and software tools*, Atmospheric Environment, 37 (2003), pp. 5083–5096.

[22] A. Sandu and P. Miehe, *Forward, tangent linear, and adjoint Runge–Kutta methods in KPP–*2.2 *for efficient chemical kinetic simulations*, Int. J. Comput. Math., 87 (2010), pp. 2458–2479.

[23] A. Sandu, J. G. Verwer, J. G. Blom, E. J. Spee, G. R. Carmichael, and F. A. Potra, *Benchmarking stiff ODE solvers for atmospheric chemistry problems* II: *Rosenbrock methods*, Atmospheric Environment, 31 (1997), pp. 3459–3472.

[24] R. Serban and A. C. Hindmarsh, *CVODES, the Sensitivity-Enabled ODE Solver in SUNDIALS*, Tech. report UCRL-PROC-210300, Lawrence Livermore National Laboratory, Livermore, CA, 2003.

[25] Z. Sirkes and E. Tziperman, *Finite difference of adjoint or adjoint of finite difference*, Monthly Weather Review, 125 (1997), pp. 3373–3378.

[26] J. G. Verwer, E. J. Spee, J. G. Blom, and W. Hunsdorfer, *A second-order Rosenbrock method applied to photochemical dispersion problems*, SIAM J. Sci. Comput., 20 (1999), pp. 1456–1480.

[27] H. Zhang and A. Sandu, *FATODE: A Library for Forward, Adjoint, and Tangent Linear Integration of ODEs*, Tech. report TR-11-25, Computer Science, Virginia Tech., Blacksburg, VA, 2011, http://eprints.cs.vt.edu/archive/00001170/01/fatode_technical_report.pdf.

[28] H. Zhang and A. Sandu, *FATODE: A library for forward, adjoint and tangent linear integration of stiff systems*, in Proceedings of Spring Simulation Multi-conference (SpringSim) 2011, Vol. High Performance Computing Symposium, L. T. Watson, G. W. Howell, W. I. Thacker, and S. Seidel, eds., SCS, Boston, MA, 2011, pp. 143–150.

[29] H. Zhang and A. Sandu, *FATODE: User's Guide*, available online from http://people.cs.vt.edu/~asandu/Software/FATODE/FATODE_user_guide.pdf (2011).

[30] H. Zhang and A. Sandu, *FATODE Website: Forward, Adjoint, and Tangent Linear Integration of ODEs*, http://people.cs.vt.edu/~asandu/Software/FATODE (2011).

[31] H. Zhang and A. Sandu, *Supplementary material for "FATODE: A library for forward, adjoint, and tangent linear integration of ODEs,"* SIAM J. Sci. Comput., 36 (2014), pp. C504–C523.