

SUPPLEMENTARY MATERIAL FOR
 “FATODE: A LIBRARY FOR FORWARD, ADJOINT, AND
 TANGENT LINEAR INTEGRATION OF ODES”

Hong Zhang and Adrian Sandu

Computational Science Laboratory,
 Department of Computer Science,
 Virginia Polytechnic Institute and State University,
 Blacksburg, VA 24061
 ({zhang,sandu}@cs.vt.edu)

Supplementary material part A. Derivatives. Consider the following smooth function

$$\phi(y, p) : \mathbb{R}^{d+m} \rightarrow \mathbb{R}^n$$

and its first and second order derivatives

$$\begin{aligned} \phi_{\mathbf{y}} &= \left(\frac{\partial \phi_i}{\partial y_j} \right)_{1 \leq i \leq n, 1 \leq j \leq d}, & \phi_{\mathbf{p}} &= \left(\frac{\partial \phi_i}{\partial p_j} \right)_{1 \leq i \leq n, 1 \leq j \leq m}, \\ \phi_{\mathbf{y}, \mathbf{y}} &= \left(\frac{\partial^2 \phi_i}{\partial y_j \partial y_k} \right)_{1 \leq i \leq n, 1 \leq j, k \leq d}, & \phi_{\mathbf{y}, \mathbf{p}} &= \left(\frac{\partial^2 \phi_i}{\partial y_j \partial p_k} \right)_{1 \leq i \leq n, 1 \leq j \leq d, 1 \leq k \leq m}, \\ \phi_{\mathbf{p}, \mathbf{p}} &= \left(\frac{\partial^2 \phi_i}{\partial p_j \partial p_k} \right)_{1 \leq i \leq n, 1 \leq j, k \leq m}, & \phi_{\mathbf{p}, \mathbf{y}} &= \left(\frac{\partial^2 \phi_i}{\partial p_j \partial y_k} \right)_{1 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq d}. \end{aligned}$$

Let $u, v \in \mathbb{R}^d$, $\bar{u}, \bar{v} \in \mathbb{R}^m$, and $z \in \mathbb{R}^n$. Hessian-vector products are matrices of the form

$$\phi_{\mathbf{y}, \mathbf{y}} \cdot u = \left(\sum_{k=1}^d \frac{\partial^2 \phi_i}{\partial y_j \partial y_k} u_k \right)_{1 \leq i \leq n, 1 \leq j \leq d}, \quad z \cdot \phi_{\mathbf{y}, \mathbf{y}} = \left(\sum_{i=1}^n \frac{\partial^2 \phi_i}{\partial y_j \partial y_k} z_i \right)_{1 \leq j, k \leq d},$$

and similar for all other derivatives. The derivatives of Jacobian-vector products are:

$$\begin{aligned} \left(\frac{d}{dy} (\phi_{\mathbf{y}} \cdot u) \right) \cdot v &= \left(\sum_{k=1}^d \frac{d}{dy_k} \left(\sum_{i=1}^d \frac{\partial \phi_j}{\partial y_i} u_i \right) v_k \right)_{1 \leq j \leq n} \\ &= \left(\sum_{i, k=1}^d \frac{\partial^2 \phi_j}{\partial y_i \partial y_k} u_i v_k \right)_{1 \leq j \leq n} \\ &= (\phi_{\mathbf{y}, \mathbf{y}} \cdot u) \cdot v \in \mathbb{R}^n \\ &= (\phi_{\mathbf{y}, \mathbf{y}} \cdot v) \cdot u \in \mathbb{R}^n. \end{aligned}$$

$$\left(\frac{d}{dy} (\phi_{\mathbf{y}}^T \cdot z) \right) \cdot u = \left(\sum_{k=1}^d \frac{d}{dy_k} \left(\sum_{i=1}^n \frac{\partial \phi_i}{\partial y_j} z_i \right) u_k \right)_{1 \leq j \leq d}$$

$$\begin{aligned}
&= \left(\sum_{k=1}^d \sum_{i=1}^n \frac{\partial^2 \phi_i}{\partial y_j \partial y_k} z_i u_k \right)_{1 \leq j \leq d} \\
&= (z \cdot \phi_{\mathbf{y}, \mathbf{y}}) \cdot u \in \mathbb{R}^d \\
&= (\phi_{\mathbf{y}, \mathbf{y}} \cdot u)^T \cdot z \in \mathbb{R}^d.
\end{aligned}$$

The derivatives of the Jacobian with respect to model parameters are

$$\begin{aligned}
\left(\frac{d}{dy} (\phi_{\mathbf{p}} \cdot \bar{u}) \right) \cdot u &= \left(\sum_{k=1}^d \frac{d}{dy_k} \left(\sum_{i=1}^m \frac{\partial \phi_j}{\partial p_i} \bar{u}_i \right) u_k \right)_{1 \leq j \leq n} \\
&= \left(\sum_{i=1}^m \sum_{k=1}^d \frac{\partial^2 \phi_j}{\partial p_i \partial y_k} u_k \bar{u}_i \right)_{1 \leq j \leq n} \\
&= (\phi_{\mathbf{p}, \mathbf{y}} \cdot u) \cdot \bar{u} \in \mathbb{R}^n \\
&= (\phi_{\mathbf{y}, \mathbf{p}} \cdot \bar{u}) \cdot u \in \mathbb{R}^n.
\end{aligned}$$

$$\begin{aligned}
\left(\frac{d}{dy} (\phi_{\mathbf{p}}^T \cdot z) \right) \cdot u &= \left(\sum_{k=1}^d \frac{d}{dy_k} \left(\sum_{i=1}^n \frac{\partial \phi_i}{\partial p_j} z_i \right) u_k \right)_{1 \leq j \leq m} \\
&= \left(\sum_{i=1}^n \sum_{k=1}^d \frac{\partial^2 \phi_i}{\partial p_j \partial y_k} u_k z_i \right)_{1 \leq j \leq m} \\
&= (z \cdot \phi_{\mathbf{p}, \mathbf{y}}) \cdot u \in \mathbb{R}^m \\
&= (\phi_{\mathbf{p}, \mathbf{y}} \cdot u)^T \cdot z \in \mathbb{R}^m.
\end{aligned}$$

$$\begin{aligned}
\left(\frac{d}{dp} (\phi_{\mathbf{y}} \cdot u) \right) \cdot \bar{u} &= \left(\sum_{k=1}^m \frac{d}{dp_k} \left(\sum_{i=1}^d \frac{\partial \phi_j}{\partial y_i} u_i \right) \bar{u}_k \right)_{1 \leq j \leq n} \\
&= \left(\sum_{i=1}^d \sum_{k=1}^m \frac{\partial^2 \phi_j}{\partial p_k \partial y_i} u_i \bar{u}_k \right)_{1 \leq j \leq n} \\
&= (\phi_{\mathbf{p}, \mathbf{y}} \cdot u) \cdot \bar{u} \in \mathbb{R}^n \\
&= (\phi_{\mathbf{y}, \mathbf{p}} \cdot \bar{u}) \cdot u \in \mathbb{R}^n.
\end{aligned}$$

$$\begin{aligned}
\left(\frac{d}{dp} (\phi_{\mathbf{y}}^T \cdot z) \right) \cdot \bar{u} &= \left(\sum_{k=1}^m \frac{d}{dp_k} \left(\sum_{i=1}^n \frac{\partial \phi_i}{\partial y_j} z_i \right) \bar{u}_k \right)_{1 \leq j \leq d} \\
&= \left(\sum_{i=1}^n \sum_{k=1}^m \frac{\partial^2 \phi_i}{\partial p_k \partial y_j} \bar{u}_k z_i \right)_{1 \leq j \leq d} \\
&= (z \cdot \phi_{\mathbf{y}, \mathbf{p}}) \cdot \bar{u} \in \mathbb{R}^d \\
&= (z \cdot \phi_{\mathbf{p}, \mathbf{y}})^T \cdot \bar{u} \in \mathbb{R}^d \\
&= (\phi_{\mathbf{y}, \mathbf{p}} \cdot \bar{u})^T \cdot z \in \mathbb{R}^d.
\end{aligned}$$

They are used to express derivatives of Jacobian-vector products as follows:

$$\left(\frac{d}{dy} (\phi_{\mathbf{y}} \cdot u) \right) \cdot v = (\phi_{\mathbf{y}, \mathbf{y}} \cdot u) \cdot v = (\phi_{\mathbf{y}, \mathbf{y}} \cdot v) \cdot u \in \mathbb{R}^n,$$

$$\begin{aligned}
\left(\frac{d}{dy}(\phi_{\mathbf{y}}^T \cdot z)\right) \cdot u &= (z \cdot \phi_{\mathbf{y},\mathbf{y}}) \cdot u = (\phi_{\mathbf{y},\mathbf{y}} \cdot u)^T \cdot z \in \mathbb{R}^d, \\
\left(\frac{d}{dp}(\phi_{\mathbf{p}} \cdot \bar{u})\right) \cdot \bar{v} &= (\phi_{\mathbf{p},\mathbf{p}} \cdot \bar{u}) \cdot \bar{v} = (\phi_{\mathbf{p},\mathbf{p}} \cdot \bar{v}) \cdot \bar{u} \in \mathbb{R}^n, \\
\left(\frac{d}{dp}(\phi_{\mathbf{p}}^T \cdot z)\right) \cdot \bar{u} &= (z \cdot \phi_{\mathbf{p},\mathbf{p}}) \cdot \bar{u} = (\phi_{\mathbf{p},\mathbf{p}} \cdot \bar{u})^T \cdot z \in \mathbb{R}^m, \\
\left(\frac{d}{dy}(\phi_{\mathbf{p}} \cdot \bar{u})\right) \cdot u &= (\phi_{\mathbf{p},\mathbf{y}} \cdot u) \cdot \bar{u} = (\phi_{\mathbf{y},\mathbf{p}} \cdot \bar{u}) \cdot u \in \mathbb{R}^n, \\
\left(\frac{d}{dy}(\phi_{\mathbf{p}}^T \cdot z)\right) \cdot u &= (\phi_{\mathbf{p},\mathbf{y}} \cdot u)^T \cdot z \in \mathbb{R}^m, \\
\left(\frac{d}{dp}(\phi_{\mathbf{y}} \cdot u)\right) \cdot \bar{u} &= (\phi_{\mathbf{y},\mathbf{p}} \cdot \bar{u}) \cdot u \in \mathbb{R}^n, \\
\left(\frac{d}{dp}(\phi_{\mathbf{y}}^T \cdot z)\right) \cdot \bar{u} &= (\phi_{\mathbf{y},\mathbf{p}} \cdot \bar{u})^T \cdot z \in \mathbb{R}^d.
\end{aligned}$$

Consider now the extended ODE (2.7). The right hand side function has the following extended Jacobian

$$\begin{aligned}
\text{(A.1)} \quad \frac{d[f, 0, r]}{d[y, p, q]} &= \mathbf{J}(t, y, p) = \begin{bmatrix} \mathbf{f}_{\mathbf{y}}(t, y, p) & \mathbf{f}_{\mathbf{p}}(t, y, p) & \mathbf{0}_{(d,1)} \\ \mathbf{0}_{(m,d)} & \mathbf{0}_{(m,m)} & \mathbf{0}_{(m,1)} \\ \mathbf{r}_{\mathbf{y}}(t, y, p) & \mathbf{r}_{\mathbf{p}}(t, y, p) & 0_{(1,1)} \end{bmatrix}, \\
\mathbf{J}^T(t, y, p) &= \begin{bmatrix} \mathbf{f}_{\mathbf{y}}^T(t, y, p) & \mathbf{0}_{(d,m)} & \mathbf{r}_{\mathbf{y}}^T(t, y, p) \\ \mathbf{f}_{\mathbf{p}}^T(t, y, p) & \mathbf{0}_{(m,m)} & \mathbf{r}_{\mathbf{p}}^T(t, y, p) \\ \mathbf{0}_{(1,d)} & \mathbf{0}_{(1,m)} & 0_{(1,1)} \end{bmatrix}.
\end{aligned}$$

The extended Hessian times vector terms reads:

$$\begin{aligned}
\text{(A.2)} \quad \left(\begin{bmatrix} u \\ \bar{u} \\ \hat{u} \end{bmatrix} \cdot \tilde{H} \right) \cdot \begin{bmatrix} v \\ \bar{v} \\ \hat{v} \end{bmatrix} &= \frac{d}{d[y, p, q]} \left(\begin{bmatrix} \mathbf{f}_{\mathbf{y}}^T & \mathbf{0} & \mathbf{r}_{\mathbf{y}}^T \\ \mathbf{f}_{\mathbf{p}}^T & \mathbf{0} & \mathbf{r}_{\mathbf{p}}^T \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} u \\ \bar{u} \\ \hat{u} \end{bmatrix} \right) \cdot \begin{bmatrix} v \\ \bar{v} \\ \hat{v} \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{f}_{\mathbf{y},\mathbf{y}} \cdot v)^T \cdot u + (\mathbf{f}_{\mathbf{y},\mathbf{p}} \cdot \bar{v})^T \cdot u + (\mathbf{r}_{\mathbf{y},\mathbf{y}} \cdot v)^T \cdot \hat{u} + (\mathbf{r}_{\mathbf{y},\mathbf{p}} \cdot \bar{v})^T \cdot \hat{u} \\ (\mathbf{f}_{\mathbf{p},\mathbf{y}} \cdot v)^T \cdot u + (\mathbf{f}_{\mathbf{p},\mathbf{p}} \cdot \bar{v})^T \cdot u + (\mathbf{r}_{\mathbf{p},\mathbf{y}} \cdot v)^T \cdot \hat{u} + (\mathbf{r}_{\mathbf{p},\mathbf{p}} \cdot \bar{v})^T \cdot \hat{u} \\ \mathbf{0} \end{bmatrix}.
\end{aligned}$$

Supplementary material part B. FATODE implementation of forward model integration.

FATODE provides a high quality solvers for the forward ODE problem (1.1). Even without the sensitivity analysis capabilities it can be used as a generic ODE solution library. In this section we discuss the efficient implementation of different Runge-Kutta and Rosenbrock methods for solving (1.1). The implementation of forward solvers is inspired by [13, 14].

B.1. Explicit Runge-Kutta methods. The implementation of explicit methods is based on (2.2). The stage vectors are computed in succession using

$$(B.1) \quad Y_1 = y_n; \quad Y_i = y_n + h \sum_{j=1}^{i-1} a_{i,j} f(T_j, Y_j), \quad i = 2, \dots, s.$$

B.2. Matrices for solving implicit methods. The implementations of implicit methods use the following matrices

$$(B.2a) \quad \mathbf{R}(\gamma, t, y) = \mathbf{I}_{(d,d)} - h \gamma \mathbf{f}_y(t, y),$$

$$(B.2b) \quad \tilde{\mathbf{R}}(\gamma, t, y) = \frac{1}{h \gamma} \mathbf{I}_{(d,d)} - \mathbf{f}_y(t, y),$$

$$(B.2c) \quad \hat{\mathbf{R}}_n = \begin{bmatrix} 1 - h a_{1,1} \mathbf{f}_y(T_1, Y_1) & \cdots & -h a_{1,s} \mathbf{f}_y(T_s, Y_s) \\ \vdots & \ddots & \vdots \\ -h a_{s,1} \mathbf{f}_y(T_1, Y_1) & \cdots & 1 - h a_{s,s} \mathbf{f}_y(T_s, Y_s) \end{bmatrix} \in \mathbb{R}^{ds \times ds}.$$

Replacing each $\mathbf{f}_y(T_i, Y_i)$ in (B.2c) by $\mathbf{f}_y(t_n, t_n)$ leads to the approximation:

$$(B.2d) \quad \bar{\mathbf{R}}_n = \mathbf{I}_{(ds,ds)} - h \mathbf{A} \otimes \mathbf{f}_y(t_n, y_n) \approx \hat{\mathbf{R}}_n,$$

where \otimes is the matrix Kronecker product [14].

B.3. Implicit Runge-Kutta methods. To reduce the influence of round-off errors, we apply the transformation $z_i = Y_i - y_n$ [14] in the formulas (2.2) to obtain

$$(B.3a) \quad T_i = t_n + c_i h, \quad z_i = h \sum_{j=1}^s a_{i,j} f(T_j, y_n + z_j), \quad i = 1, \dots, s,$$

$$(B.3b) \quad y_{n+1} = y_n + \sum_{i=1}^s d_i z_i.$$

The new coefficients are

$$(B.4) \quad \mathbf{d} = [d_i]_{1 \leq i \leq s}, \quad \mathbf{d}^T = \mathbf{b}^T \cdot \mathbf{A}^{-1}.$$

B.4. Singly diagonally implicit Runge-Kutta methods. The stage equations (B.3a) read

$$(B.5) \quad z_i = h \sum_{j=1}^{i-1} a_{i,j} f(T_j, y_n + z_j) + h \gamma f(T_i, y_n + z_i).$$

The nonlinear systems of equations (B.5) are solved in succession for each stage $i = 1, \dots, s$ by simplified Newton iterations of the form

$$(B.6) \quad \begin{aligned} \mathbf{R}(\gamma, t_n, y_n) \cdot \Delta z_i^{[k]} &= z_i^{[k]} - h \sum_{j=1}^{i-1} a_{i,j} f(T_j, y_n + z_j) \\ z_i^{[k+1]} &= z_i^{[k]} - \Delta z_i^{[k]}, \quad k = 0, 1, \dots \end{aligned}$$

The same matrix is shared for all iterations and all stages, so that only one LU decomposition of \mathbf{R} is performed in each time step.

B.5. Fully implicit Runge-Kutta methods. Fully implicit Runge-Kutta methods require the solution of the $ds \times ds$ nonlinear system (B.3a) [22]. With the compact notation

$$(B.7) \quad Z = [z_1^T \dots z_s^T]^T, \quad F(Z) = [f^T(T_1, y_n + z_1) \dots f^T(T_s, y_n + z_s)]^T,$$

where $Z, F(Z) \in \mathbb{R}^{ds}$, the nonlinear system (B.3a) can be written as

$$(B.8) \quad Z = (\mathbf{A} \otimes \mathbf{I}_{(d,d)}) \cdot F(Z).$$

The system (B.8) is solved by simplified Newton iterations [14],

$$(B.9) \quad \begin{aligned} \bar{\mathbf{R}}_n \cdot \Delta Z^{[k]} &= Z^{[k]} - (h \mathbf{A} \otimes \mathbf{I}_{(d,d)}) \cdot F(Z^{[k]}) \\ Z^{[k+1]} &= Z^{[k]} - \Delta Z^{[k]}, \quad k = 0, 1, \dots \end{aligned}$$

Note that only the Jacobian at the beginning of the time step is used in Newton's iterations. Following [14], our implementation of the fully implicit s -stage Runge-Kutta method uses a transformation of the system (B.9) to a complex form such that the costly ds -dimensional real LU decomposition is replaced by d -dimensional LU decompositions of matrices of the form $\mathbf{R}(\lambda_i, t_n, y_n)$, where λ_i are the eigenvalues of \mathbf{A} . For the 3-stage methods implemented in FATODE the coefficient matrices \mathbf{A} have one real and two complex conjugate eigenvalues, which leads to solving one real and one complex d -dimensional systems.

B.6. Rosenbrock methods. For implementation purpose, we use the alternative formulation [13] of the formula (2.4)

$$(B.10a) \quad T_i = t_n + \alpha_i h, \quad Y_i = y_n + \sum_{j=1}^{i-1} \alpha_{i,j} k_j,$$

$$(B.10b) \quad \tilde{\mathbf{R}}(\gamma, t_n, y_n) \cdot k_i = f(T_i, Y_i) + \sum_{j=1}^{i-1} \frac{c_{i,j}}{h} k_j + h \gamma_i f_t(t_n, y_n),$$

$$(B.10c) \quad y_{n+1} = y_n + \sum_{i=1}^s m_i k_i,$$

where $\mathbf{a} = [a_{i,j}]_{1 \leq i, j \leq s}$, $\mathbf{c} = [c_{i,j}]_{1 \leq i, j \leq s}$, $\mathbf{m} = [m_i]_{1 \leq i \leq s}$, are defined by

$$(B.11) \quad \mathbf{a} = \boldsymbol{\alpha} \cdot \boldsymbol{\gamma}^{-1}, \quad \mathbf{c} = \text{diag}(\boldsymbol{\gamma}^{-1}) - \boldsymbol{\gamma}^{-1}, \quad \mathbf{m} = \boldsymbol{\gamma}^{-T} \cdot \mathbf{b}.$$

At each stage (B.10b) the solution of a linear system of dimension $d \times d$ is required. The same matrix $\tilde{\mathbf{R}}$ is shared by all the stages and one LU decomposition per step is required.

Supplementary material part C. FATODE implementation of tangent linear model integration.

Tangent linear models are derived for direct sensitivity analysis with each of the families of methods in FATODE. Highly efficient implementations are obtained by re-using the LU decompositions from the forward solution on the sensitivity equations [8].

C.1. Tangent linear Runge-Kutta methods. A tangent linear Runge-Kutta (2.2) method reads

$$(C.1a) \quad Y_i = y_n + h \sum_{j=1}^s a_{i,j} f(T_j, Y_j), \quad \dot{Y}_i = \dot{y}_n + h \sum_{j=1}^s a_{i,j} \mathbf{f}_y(T_i, Y_i) \cdot \dot{Y}_i,$$

$$(C.1b) \quad y_{n+1} = y_n + h \sum_{i=1}^s b_i f(T_i, Y_i), \quad \dot{y}_{n+1} = \dot{y}_n + h \sum_{i=1}^s b_i \mathbf{f}_y(T_i, Y_i) \cdot \dot{Y}_i.$$

Similar to the implementation of implicit forward integrators, we introduce the sensitivity stage variables $\dot{z}_i = \dot{Y}_i - \dot{y}_n$ and the sensitivity part becomes

$$(C.2a) \quad \dot{z}_i - h \sum_{j=1}^s a_{i,j} \mathbf{f}_y(T_j, Y_j) \cdot \dot{z}_j = h \sum_{j=1}^s a_{i,j} \mathbf{f}_y(T_j, Y_j) \cdot \dot{y}_n, \quad i = 1, \dots, s,$$

$$(C.2b) \quad \dot{y}_{n+1} = \dot{y}_n + \sum_{i=1}^s d_i \dot{z}_i.$$

Using the compact notation (B.7) and the matrix (B.2c) the stage equations (C.2a) can be written as

$$(C.3) \quad \widehat{\mathbf{R}}_n \cdot \dot{Z} = \left(\mathbf{I}_{(sd, sd)} - \widehat{\mathbf{R}}_n \right) \cdot (\mathbf{1}_s \otimes \dot{y}_n).$$

C.2. Explicit RK methods. For ERK methods the equations (C.1a) are solved successively for each stage $i = 1, \dots, s$, using

$$\dot{Y}_1 = \dot{y}_n; \quad \dot{Y}_i = \dot{y}_n + h \sum_{j=1}^{i-1} a_{i,j} \mathbf{f}_y(T_j, Y_j) \cdot \dot{Y}_j, \quad i = 2, \dots, s.$$

C.3. Singly diagonally implicit Runge-Kutta methods. For SDIRK methods the system (C.2a) reduces to s independent d -dimensional linear systems that are solved successively for each stage $i = 1, \dots, s$

$$\mathbf{R}(\gamma, T_i, Y_i) \cdot \dot{z}_i = h \sum_{j=1}^{i-1} a_{i,j} \mathbf{f}_y(T_j, Y_j) \cdot (\dot{y}_n + \dot{z}_j) + h \gamma \mathbf{f}_y(T_i, Y_i) \cdot \dot{y}_n.$$

FATODE allows users to choose to solve the linear system (C.4) directly at the expense of an additional LU decomposition of the matrix $\mathbf{R}(\gamma, T_i, Y_i)$ per stage, or to apply simplified Newton iterations of the form

$$(C.4) \quad \begin{aligned} \mathbf{R}(\gamma, t_n, y_n) \cdot \Delta \dot{z}_i^{[m]} &= \dot{z}_i^{[m]} - h \sum_{j=1}^i a_{i,j} \mathbf{f}_y(T_j, Y_j) \cdot (\dot{y}_n + \dot{z}_j^{[m]}) \\ \dot{z}_i^{[m+1]} &= \dot{z}_i^{[m]} - \Delta \dot{z}_i^{[m]}, \quad m = 0, 1, \dots \end{aligned}$$

The LU decomposition of the matrix $\mathbf{R}(\gamma, t_n, y_n)$ is also necessary in forward integration. Equations (C.4) re-use the LU decomposition which is available after the equations (B.6) are calculated in each step. FATODE controls the iteration number, and possibly the step size, such that the iteration error in (C.4) is smaller than the local truncation error at the current step. When (C.4) is used the accuracy of the sensitivity coefficients is of the same order as the local truncation error. The reuse of the forward LU factorization can save considerable CPU time.

C.4. Fully implicit Runge-Kutta methods. For fully implicit Runge-Kutta methods two options are available for solving the system (C.3). One is to construct the $ds \times ds$ linear system (C.3) explicitly and solve it directly by factorizing the matrix $\widehat{\mathbf{R}}_n$.

The other is to apply simplified Newton iterations of the form

$$\begin{aligned} \overline{\mathbf{R}}_n \cdot \Delta \dot{Z}^{[m]} &= \widehat{\mathbf{R}}_n \cdot (\mathbf{1}_s \otimes \dot{y}_n + \dot{Z}^{[m]}) - \mathbf{1}_s \otimes \dot{y}_n \\ \dot{Z}^{[m+1]} &= \dot{Z}^{[m]} - \Delta \dot{Z}^{[m]}, \quad m = 0, 1, \dots \end{aligned} \quad (\text{C.5})$$

The matrix $\overline{\mathbf{R}}_n$ of the resulting $ds \times ds$ linear system is available from the forward solution process, i.e., the calculations of the equations (B.9). The real and complex LU decompositions can be reused. According to our experience, the second option is usually more efficient than the first one for large systems.

C.5. Rosenbrock methods. The tangent linear Rosenbrock method consists of the formula formula (B.10) plus the sensitivity part, which is obtained by differentiating the formula (B.10). In each step we solve the combined set of equations

$$\widetilde{\mathbf{R}}(h\gamma, t_n, y_n) \cdot k_i = f(T_i, Y_i) + \sum_{j=1}^{i-1} \frac{c_{i,j}}{h} k_j + h\gamma_i k_i f_t(t_n, y_n), \quad (\text{C.6a})$$

$$\begin{aligned} \widetilde{\mathbf{R}}(h\gamma, t_n, y_n) \cdot \dot{k}_i &= \mathbf{f}_{\mathbf{y}}(T_i, Y_i) \cdot \left(\dot{y}_n + \sum_{j=1}^{i-1} a_{i,j} \dot{k}_j \right) + \sum_{j=1}^{i-1} \frac{c_{i,j}}{h} \dot{k}_j \\ &\quad + (\dot{y}_n \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}(t_n, y_n)) \cdot k_i + h\gamma_i f_{y,t}(t_n, y_n) \cdot \dot{y}_n, \end{aligned} \quad (\text{C.6b})$$

$$y_{n+1} = y_n + \sum_{i=1}^s m_i k_i, \quad (\text{C.6c})$$

$$\dot{y}_{n+1} = \dot{y}_n + \sum_{i=1}^s m_i \dot{k}_i. \quad (\text{C.6d})$$

The stage vectors \dot{k}_i are obtained in succession by solving a sequence of linear systems, all of which re-use the LU decomposition of $\widetilde{\mathbf{R}}(h\gamma, t_n, y_n)$ performed in (B.10). Formula (C.6b) involves the Hessian tensor $\mathbf{f}_{\mathbf{y},\mathbf{y}}(t_n, y_n)$. In practice, an analytical Hessian tensor is difficult to obtain, and its evaluation is costly in both CPU time and memory storage. Note that the above equation only needs the product $(\dot{y}_n \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}(t_n, y_n)) \cdot k_i$. Such terms can be obtained efficiently using automatic differentiation [4, 11] twice, in forward over reverse mode.

Supplementary material part D. FATODE implementation of discrete adjoint model integration: sensitivities with respect to initial conditions.

FATODE implements discrete adjoints of all the methods. Such discrete adjoints have good theoretical properties, in the sense that they are consistent discretizations of the adjoint ODE [19, 20]. This section discusses the adjoint sensitivities with respect to initial conditions.

The discrete adjoint Runge-Kutta method [12] solving the discrete adjoint equations (2.15) reads

$$(D.1a) \quad u_i = h \mathbf{f}_y^T(T_i, Y_i) \cdot \left(b_i \lambda_{n+1} + \sum_{j=1}^s a_{j,i} u_j \right), \quad i = s, \dots, 1,$$

$$(D.1b) \quad \lambda_n = \lambda_{n+1} + \sum_{j=1}^s u_j.$$

The stage equations (D.1a) form a $ds \times ds$ linear system involving the transpose of matrix (B.2c):

$$(D.2) \quad \begin{aligned} U &= [u_1^T \dots u_s^T]^T, \\ \widehat{\mathbf{R}}_n^T \cdot U &= h [b_1 \lambda_{n+1}^T \mathbf{f}_y(T_1, Y_1) \dots b_s \lambda_{n+1}^T \mathbf{f}_y(T_s, Y_s)]^T. \end{aligned}$$

For $b_i \neq 0$ one can rewrite (D.1) as another Runge-Kutta method [12] applied to the adjoint ODE

$$(D.3) \quad \begin{aligned} \ell_i &= \mathbf{f}_y^T(t_{n+1} - \bar{c}_i h, Y_{s+1-i}) \cdot \left(\lambda^{n+1} + h \sum_{j=1}^s \bar{a}_{i,j} \ell_j \right), \quad i = s, \dots, 1, \\ \lambda^n &= \lambda^{n+1} + h \sum_{i=1}^s \bar{b}_i \ell_i, \\ \text{where} \quad \bar{b}_i &= b_{s+1-i}, \quad \bar{c}_i = 1 - c_{s+1-i}, \quad \bar{a}_{i,j} = \frac{a_{s+1-j, s+1-i} \cdot b_{s+1-j}}{b_{s+1-i}}. \end{aligned}$$

The method $(\bar{\mathbf{A}}, \bar{b}, \bar{c})$ is called the *formal adjoint* of the Runge-Kutta method (\mathbf{A}, b, c) [19]. The formal adjoint of Radau-2A is Radau-1A, vice versa. Lobatto-3C, Gauss and SDIRK-3a are formally self-adjoint.

D.1. Explicit Runge-Kutta methods. The stage equations (D.1a) are solved in succession for stages s down to 1:

$$(D.4) \quad \begin{aligned} u_s &= h b_s \mathbf{f}_y^T(T_s, Y_s) \lambda^{n+1}, \\ u_i &= h \mathbf{f}_y^T(T_i, Y_i) \cdot \left(b_i \lambda^{n+1} + \sum_{j=i+1}^s a_{j,i} u_j \right), \quad i = s-1, \dots, 1. \end{aligned}$$

Each stage i requires the computation of the Jacobian $\mathbf{f}_y(T_i, Y_i)$, forming the vector $b_i \lambda^{n+1} + \sum_{j=i+1}^s a_{j,i} u_j$ from previously computed stages $u_s \dots u_{i+1}$, and performing one Jacobian vector product.

D.2. Singly diagonally implicit Runge-Kutta methods. For SDIRK methods the s stages of the system (D.1a) are solved successively from the last stage to the first. Each stage requires the solution of a different linear system:

$$(D.5) \quad \mathbf{R}(\gamma, T_i, Y_i) \cdot u_i = h \mathbf{f}_{\mathbf{y}}^T(T_i, Y_i) \cdot \left(b_i \lambda^{n+1} + \sum_{j=i+1}^s a_{j,i} u_j \right), \quad i = s, \dots, 1.$$

FATODE offers two options: to form and solve one linear system (D.5) per stage, or to employ simplified Newton iterations of the form (C.4) and re-use the LU decomposition of $\mathbf{R}(\gamma, t_n, y_n)$ for all stages. When the iterative approach is used the accuracy of the gradients is of the same order as the local truncation error. Considerable CPU time can be saved if checkpointing the forward LU factorization is feasible from a storage perspective.

D.3. Fully implicit Runge-Kutta methods. For the fully implicit Runge-Kutta methods the $ds \times ds$ system (D.2) is fully coupled. FATODE offers two approaches to solve it. The first is to build and solve directly (D.2) via a $ds \times ds$ LU decomposition of $\widehat{\mathbf{R}}_n$. The second approach uses simplified Newton iterations of the form (C.3), where $\widehat{\mathbf{R}}_n$ is replaced by $\overline{\mathbf{R}}_n$ in (D.2), and the transformation to real and complex systems is performed. The real and complex LU factorizations associated with the matrix $\overline{\mathbf{R}}_n$ are re-used in all iterations. These factorizations are computed during the forward solution, and in principle they can be checkpointed. The tradeoff between the size of the LU factorizations to store and the time needed to recompute them will determine the best strategy.

D.4. Rosenbrock methods. The discrete Rosenbrock adjoint [1] reads

$$(D.6a) \quad \widetilde{\mathbf{R}}^T(h\gamma, t_n, y_n) \cdot u_i = m_i \lambda_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} v_j + \frac{c_{j,i}}{h} u_j \right),$$

$$(D.6b) \quad v_i = \mathbf{f}_{\mathbf{y}}^T(T_i, Y_i) \cdot u_i, \quad i = s, \dots, 1,$$

$$(D.6c) \quad \lambda_n = \lambda_{n+1} + \sum_{i=1}^s (u_i \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}(t_n, y_n)) \cdot k_i + h \mathbf{f}_{\mathbf{y},\mathbf{t}}^T(t_n, y_n) \cdot \sum_{i=1}^s \gamma_i u_i + \sum_{i=1}^s v_i.$$

The linear system (D.6a) can be solved directly at each stage. Users have to supply a routine for calculating the term $(u_i \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}(t_n, y_n)) \cdot k_i$, whose meaning is explained in Appendix A. Automatic differentiation tools like TAMC [11] provide considerable help: the product between the Hessian transposed times vector can be obtained by two consecutive runs of TAMC in forward mode.

Supplementary material part E. Discrete adjoint sensitivities with respect to parameters: general approach. Consider a numerical solution of (2.7):

$$(E.1) \quad \begin{aligned} y_{n+1} &= \Phi^n(y_n, p_n), \\ p_{n+1} &= p_n, \\ q_{n+1} &= q_n + \Omega^n(y_n, p_n), \quad n = 0, \dots, N-1, \end{aligned}$$

together with the numerical evaluation of the cost function (2.8)

$$(E.2) \quad \Psi = g(y_N, p) + q_N.$$

Replacing $p_n = p$ for all n in (E.2) leads to the *discrete forward model*:

$$(E.3) \quad y_{n+1} = \Phi^n(y_n, p); \quad q_{n+1} = q_n + \Omega^n(y_n, p), \quad n = 0, \dots, N-1.$$

Differentiating (E.3) in the direction \dot{p} yields the *discrete tangent linear model*:

$$(E.4) \quad \begin{aligned} \dot{y}_0 &= 0, \quad \dot{p}_0 = \dot{p}, \quad \dot{q}_0 = 0 \\ \dot{y}_{n+1} &= \Phi_y^n(y_n, p_n) \cdot \dot{y}_n + \Phi_p^n(y_n, p_n) \cdot \dot{p}_n, \\ \dot{p}_{n+1} &= \dot{p}_n, \\ \dot{q}_{n+1} &= \dot{q}_n + \Omega_y^n(y_n, p_n) \cdot \dot{y}_n + \Omega_p^n(y_n, p_n) \cdot \dot{p}_n. \end{aligned}$$

Replacing $p_n = p$ and $\dot{p}_n = \dot{p}$ for all n leads to (E.5)

$$(E.5) \quad \begin{aligned} \dot{y}_0 &= 0, \quad \dot{q}_0 = 0, \\ \dot{y}_{n+1} &= \Phi_y^n(y_n, p) \cdot \dot{y}_n + \Phi_p^n(y_n, p) \cdot \dot{p}, \\ \dot{q}_{n+1} &= \dot{q}_n + \Omega_y^n(y_n, p) \cdot \dot{y}_n + \Omega_p^n(y_n, p) \cdot \dot{p}. \end{aligned}$$

From (E.2) we define the co-state variables at $t_N = t_F$

$$(E.6) \quad \begin{bmatrix} \lambda_N \\ \mu_N \\ \theta_N \end{bmatrix} = \left(\frac{d\Psi}{d[y_N, p_N, q_N]} \right)^T = \begin{bmatrix} g_y^T(y_N, p) \\ g_p^T(y_N, p) \\ 1 \end{bmatrix}.$$

The backwards in time evolution of the adjoint variables is governed by the discrete adjoint equations obtained by differentiating and transposing (E.2)

$$(E.7) \quad \begin{bmatrix} \lambda_n \\ \mu_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} \Phi_y^n(y_n, p) & \Phi_p^n(y_n, p) & 0 \\ 0 & \mathbf{I} & 0 \\ \Omega_y^n(y_n, p) & \Omega_p^n(y_n, p) & \mathbf{I} \end{bmatrix}^T \begin{bmatrix} \lambda_{n+1} \\ \mu_{n+1} \\ \theta_{n+1} \end{bmatrix}, \quad n = N-1, \dots, 1.$$

or

$$\begin{bmatrix} \lambda_i \\ \mu_i \\ \theta_i \end{bmatrix} = \begin{bmatrix} \Phi_y^i(y_i, p) & 0 & \Omega_y^i(y_i, p) \\ \Phi_p^i(y_i, p) & \mathbf{I} & \Omega_p^i(y_i, p) \\ 0 & 0 & \Omega_z^i \end{bmatrix} \begin{bmatrix} \lambda_{i+1} \\ \mu_{i+1} \\ \theta_{i+1} \end{bmatrix}, \quad i = N-1, \dots, 1.$$

or, equivalently,

$$\begin{aligned} \lambda_n &= (\Phi_y^n(y_n, p_n))^T \cdot \lambda_{n+1} + (\Omega_y^n(y_n, p_n))^T \cdot \theta_{n+1}, \\ \mu_n &= \mu_{n+1} + (\Phi_p^n(y_n, p))^T \cdot \lambda_{n+1} + (\Omega_p^n(y_n, p))^T \cdot \theta_{n+1}, \\ \theta_n &= \Omega_q^n(y_n, p, q_n) \theta_{n+1}. \end{aligned}$$

and finally

$$\begin{aligned}
& \lambda_n = (\Phi_y^n(y_n, p))^T \lambda_{n+1} + (\Omega_y^n(y_n, p_n))^T \theta_{n+1}, \\
\text{(E.8)} \quad & \mu_n = \mu_{n+1} + (\Phi_p^n(y_n, p_n))^T \lambda_{n+1} + (\Omega_p^n(y_n, p_n))^T \theta_{n+1}, \\
& \theta_n = \theta_{n+1}.
\end{aligned}$$

From (E.6) and (E.7) one infers that $\theta_n = 1$ for all n . Using this fact, a rearrangement of (E.7) leads to the *discrete adjoint equations*

$$\begin{aligned}
& \lambda_N = g_y^T(y_N, p), \quad \mu_N = g_p^T(y_N, p), \\
\text{(E.9)} \quad & \lambda_n = (\Phi_y^n(y_n, p))^T \cdot \lambda_{n+1} + (\Omega_y^n(y_n, p))^T, \\
& \mu_n = \mu_{n+1} + (\Phi_p^n(y_n, p))^T \cdot \lambda_{n+1} + (\Omega_p^n(y_n, p))^T, \quad n = N-1, \dots, 0.
\end{aligned}$$

The adjoint values at the initial time represent the sensitivities of the numerical cost function (E.2) with respect to the initial conditions and with parameters, respectively:

$$\text{(E.10)} \quad \left(\frac{\partial \Psi}{\partial y_0} \right)^T = \lambda_0, \quad \left(\frac{\partial \Psi}{\partial p} \right)^T = \mu_0.$$

For details on derivation see [19].

Supplementary material part F. FATODE implementation of discrete adjoint model integration: sensitivities with respect to a vector of parameters.

We now consider the case where the adjoint sensitivity is computed with respect to a time-independent vector of parameters $p \in \mathbb{R}^m$ which appears in the right hand side of (1.1). We consider a scalar quantity of interest having the general form (2.6) (with the number of outputs $o = 1$). As shown in Appendix E the numerical solution of (2.7) provides the discrete y_n and q_n . The discrete adjoint model equations calculate the adjoint variables λ_n and μ_n backward in time, such that

$$(F.1) \quad \lambda_N = \mathbf{g}_y^T(y_N, p), \quad \mu_N = \mathbf{g}_p^T(y_N, p); \quad \lambda_0 = (\partial\Psi/\partial y_0)^T, \quad \mu_0 = (\partial\Psi/\partial p)^T.$$

For details on derivation see [19] and Appendix E.

F.1. Runge-Kutta methods. Consider the Runge-Kutta method (2.2) applied to the extended ODE system (2.7)

$$(F.2a) \quad Y_i = y_n + h \sum_{j=1}^s a_{i,j} f(T_j, Y_j, p), \quad i = 1, \dots, s,$$

$$(F.2b) \quad y_{n+1} = y_n + h \sum_{j=1}^s b_j f(T_j, Y_j, p),$$

$$(F.2c) \quad q_{n+1} = q_n + h \sum_{j=1}^s b_j r(T_j, Y_j, p).$$

Note that, since r does not depend on q , there is no need to compute the stage values for the quadrature variable.

Using the extended co-state vector and the extended Jacobian (A.1), the discrete adjoint (E.7) of a Runge-Kutta method is

$$(F.3a) \quad \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = h \begin{bmatrix} \mathbf{f}_y^T(T_i, Y_i, p) & \mathbf{0}_{(d,m)} & \mathbf{r}_y^T(T_i, Y_i, p) \\ \mathbf{f}_p^T(T_i, Y_i, p) & \mathbf{0}_{(m,m)} & \mathbf{r}_p^T(T_i, Y_i, p) \\ \mathbf{0}_{(1,d)} & \mathbf{0}_{(1,m)} & 0_{(1,1)} \end{bmatrix} \cdot \left(b_i \begin{bmatrix} \lambda_{n+1} \\ \mu_{n+1} \\ \theta_{n+1} \end{bmatrix} + \sum_{j=1}^s a_{j,i} \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} \right), \quad i = s, \dots, 1,$$

$$(F.3b) \quad \begin{bmatrix} \lambda_n \\ \mu_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} \lambda_{n+1} \\ \mu_{n+1} \\ \theta_{n+1} \end{bmatrix} + \sum_{j=1}^s \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix}.$$

From the last equation of (F.3a) we see that $w_i = 0$ for all i , and from (F.3b) we infer that $\theta_n = \theta_{n+1} = \dots = \theta_N = 1$. The discrete adjoint Runge-Kutta method (F.3) can be rewritten as

$$(F.4a) \quad u_i = h \mathbf{f}_y^T(T_i, Y_i, p) \cdot \left(b_i \lambda_{n+1} + \sum_{j=1}^s a_{j,i} u_j \right) + h b_i \mathbf{r}_y^T(T_i, Y_i, p),$$

$$(F.4b) \quad v_i = h \mathbf{f}_p^T(T_i, Y_i, p) \cdot \left(b_i \lambda_{n+1} + \sum_{j=1}^s a_{j,i} u_j \right)$$

$$+h b_i \mathbf{r}_p^T(T_i, Y_i, p), \quad i = s \dots 1,$$

$$(F.4c) \quad \lambda_n = \lambda_{n+1} + \sum_{j=1}^s u_j,$$

$$(F.4d) \quad \mu_n = \mu_{n+1} + \sum_{j=1}^s v_j.$$

The stages u_i are obtained by solving the system in a similar way as solving system (D.4) and the implementation differs between SDIRK and fully implicit 3-stage Runge-Kutta method. Then v_i can be readily obtained from the right-hand side calculation.

Sensitivities with respect to initial conditions are obtained by setting all derivatives with respect to parameters to zero in (F.4), to obtain (D.1).

F.2. Rosenbrock methods. Applying the Rosenbrock method (B.10) to the extended system (2.7) gives the following formula for evaluating the quadrature term in the cost functional (2.6):

$$(F.5a) \quad \widehat{k}_i = h\gamma \left(r(T_i, Y_i) + \sum_{j=1}^{i-1} \frac{c_{i,j}}{h} \widehat{k}_j + h\gamma_i r_t + \mathbf{r}_y \cdot k_i \right),$$

$$(F.5b) \quad q_{n+1} = q_n + \sum_{i=1}^s m_i \widehat{k}_i.$$

Equation (F.5) is evaluated simultaneously with the ODE integration.

The discrete adjoint (E.7) of a Rosenbrock method is

$$\begin{aligned} & \begin{bmatrix} \frac{I_{(d,d)}}{h\gamma} - \mathbf{f}_y^T(t_n, y_n, p) & 0 & -\mathbf{r}_y^T(t_n, y_n, p) \\ -\mathbf{f}_p^T(t_n, y_n, p) & \frac{I_{(m,m)}}{h\gamma} & -\mathbf{r}_p^T(t_n, y_n, p) \\ 0 & 0 & \frac{1}{h\gamma} \end{bmatrix} \cdot \begin{bmatrix} u_i \\ \bar{u}_i \\ \widehat{u}_i \end{bmatrix} = m_i \begin{bmatrix} \lambda_{n+1} \\ \mu_{n+1} \\ \theta_{n+1} \end{bmatrix} \\ & + \sum_{j=i+1}^s \left(a_{j,i} \begin{bmatrix} v_j \\ \bar{v}_j \\ \widehat{v}_j \end{bmatrix} + \frac{c_{j,i}}{h} \begin{bmatrix} u_j \\ \bar{u}_j \\ \widehat{u}_j \end{bmatrix} \right), \\ & \begin{bmatrix} v_i \\ \bar{v}_i \\ \widehat{v}_i \end{bmatrix} = \begin{bmatrix} \mathbf{f}_y^T(T_i, Y_i, p) & 0 & \mathbf{r}_y^T(T_i, Y_i, p) \\ \mathbf{f}_p^T(T_i, Y_i, p) & 0 & \mathbf{r}_p^T(T_i, Y_i, p) \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_i \\ \bar{u}_i \\ \widehat{u}_i \end{bmatrix}, \quad i = s, s-1, \dots, 1, \\ & \begin{bmatrix} \lambda_n \\ \mu_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} \lambda_{n+1} \\ \mu_{n+1} \\ \theta_{n+1} \end{bmatrix} + \sum_{i=1}^s \left(\begin{bmatrix} u_i \\ \bar{u}_i \\ \widehat{u}_i \end{bmatrix} \cdot \widetilde{\mathbf{H}} \right) \cdot \begin{bmatrix} k_i \\ \bar{k}_i \\ \widehat{k}_i \end{bmatrix} \\ & + h \begin{bmatrix} \mathbf{f}_{y,t}^T(t_n, y_n, p) & 0 & \mathbf{r}_{y,t}^T(t_n, y_n, p) \\ \mathbf{f}_{p,t}^T(t_n, y_n, p) & 0 & \mathbf{r}_{p,t}^T(t_n, y_n, p) \\ 0 & 0 & 0 \end{bmatrix} \cdot \sum_{i=1}^s \gamma_i \begin{bmatrix} u_i \\ \bar{u}_i \\ \widehat{u}_i \end{bmatrix} + \sum_{i=1}^s \begin{bmatrix} v_i \\ \bar{v}_i \\ \widehat{v}_i \end{bmatrix}. \end{aligned}$$

Using the derivative notation in Appendix A, this equation can be written component by component as follows:

$$\frac{1}{h\gamma} \widehat{u}_i = m_i \theta_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} \widehat{v}_j + \frac{c_{j,i}}{h} \widehat{u}_j \right),$$

$$\begin{aligned}
\left(\frac{\mathbf{I}^{(d,d)}}{h\gamma} - \mathbf{f}_{\mathbf{y}}^T(t_n, y_n, p) \right) u_i &= \mathbf{r}_{\mathbf{y}}^T(t, y, p) \hat{u}_i + m_i \lambda_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} v_j + \frac{c_{j,i}}{h} u_j \right), \\
\frac{1}{h\gamma} \bar{u}_i &= \mathbf{f}_{\mathbf{p}}^T(t_n, y_n, p) u_i + \mathbf{r}_{\mathbf{p}}^T(t_n, y_n, p) \hat{u}_i + m_i \mu_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} \bar{v}_j + \frac{c_{j,i}}{h} \bar{u}_j \right), \\
v_i &= \mathbf{f}_{\mathbf{y}}^T(T_i, Y_i, p) u_i + \mathbf{r}_{\mathbf{y}}^T(T_i, Y_i, p) \hat{u}_i, \\
\bar{v}_i &= \mathbf{f}_{\mathbf{p}}^T(T_i, Y_i, p) u_i + \mathbf{r}_{\mathbf{p}}^T(T_i, Y_i, p) \hat{u}_i, \\
\hat{v}_i &= 0, \\
\lambda_n &= \lambda_{n+1} + \sum_{i=1}^s \left((u_i \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}) \cdot k_i + (u_i \cdot \mathbf{f}_{\mathbf{y},\mathbf{p}}) \cdot \bar{k}_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{y},\mathbf{y}}) \cdot k_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{y},\mathbf{p}}) \cdot \bar{k}_i \right), \\
&\quad + h \mathbf{f}_{\mathbf{y},\mathbf{t}}^T(t, y, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{y},\mathbf{t}}^T(t, y, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s v_i, \\
\mu_n &= \mu_{n+1} + \sum_{i=1}^s \left((u_i \cdot \mathbf{f}_{\mathbf{p},\mathbf{y}}) \cdot k_i + (u_i \cdot \mathbf{f}_{\mathbf{p},\mathbf{p}}) \cdot \bar{k}_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{p},\mathbf{y}}) \cdot k_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{p},\mathbf{p}}) \cdot \bar{k}_i \right), \\
&\quad + h \mathbf{f}_{\mathbf{p},\mathbf{t}}^T(t, y, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{p},\mathbf{t}}^T(t, y, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s \bar{v}_i, \\
\theta_n &= \theta_{n+1} + \sum_{i=1}^s \hat{v}_i.
\end{aligned}$$

This equation can be simplified using the following facts.

- Note that $\theta_n = 1$ for all n and $\hat{v}_i = 0$ for all i , therefore the numbers \hat{u}_i can be computed by the simple recurrence

$$(F.6) \quad \frac{1}{h\gamma} \hat{u}_i = m_i + \sum_{j=i+1}^s \left(\frac{c_{j,i}}{h} \hat{u}_j \right) \Leftrightarrow \hat{\mathbf{u}} = h\gamma (\mathbf{I}_{(s,s)} - \gamma \mathbf{c}^T)^{-1} \mathbf{m}.$$

Equations (F.6) and (B.11) lead to

$$\hat{\mathbf{u}} = h\mathbf{b}.$$

- Equation (B.10c)

$$(F.7) \quad \begin{bmatrix} y_{n+1} \\ p_{n+1} \\ q_{n+1} \end{bmatrix} = \begin{bmatrix} y_n \\ p_n \\ q_n \end{bmatrix} + \sum_{i=1}^s m_i \begin{bmatrix} k_i \\ \bar{k}_i \\ \hat{k}_i \end{bmatrix}$$

indicates that $\bar{k}_i = 0$ for all time steps since $p_{n+1} = p_n = p$. The vector k_i and \hat{k}_i are obtained from the solution of the extended form of forward integration (B.10b) and (F.5).

- The equation for calculating the quantity \bar{u} is not needed to update the adjoint variables λ and μ so that the third equation can be omitted.

With the above simplifications, and using (A.2) and the derivative notation of Appendix A, the discrete adjoint Rosenbrock method can be written in a component-wise manner as follows:

$$\frac{1}{h\gamma} \hat{u}_i = m_i + \sum_{j=i+1}^s \left(\frac{c_{j,i}}{h} \hat{u}_j \right)$$

$$\begin{aligned}
\left(\frac{\mathbf{I}^{(d,d)}}{h\gamma} - \mathbf{f}_{\mathbf{y}}^T(t_n, y_n, p)\right) u_i &= \hat{u}_i \mathbf{r}_{\mathbf{y}}^T(t_n, y_n, p) + m_i \lambda_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} v_j + \frac{c_{j,i}}{h} u_j\right) \\
v_i &= \mathbf{f}_{\mathbf{y}}^T(T_i, Y_i, p) \cdot u_i + \hat{u}_i \mathbf{r}_{\mathbf{y}}^T(T_i, Y_i, p) \\
\bar{v}_i &= \mathbf{f}_{\mathbf{p}}^T(T_i, Y_i, p) \cdot u_i + \hat{u}_i \mathbf{r}_{\mathbf{p}}^T(T_i, Y_i, p) \\
\lambda_n &= \lambda_{n+1} + \sum_{i=1}^s \left((u_i \cdot \mathbf{f}_{\mathbf{y},\mathbf{y}}) \cdot k_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{y},\mathbf{y}}) \cdot k_i \right) \\
&\quad + h \mathbf{f}_{\mathbf{y},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{y},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s v_i \\
&= \lambda_{n+1} + \sum_{i=1}^s \left((\mathbf{f}_{\mathbf{y},\mathbf{y}} \cdot k_i)^T \cdot u_i + \hat{u}_i (\mathbf{r}_{\mathbf{y},\mathbf{y}} \cdot k_i)^T \right) \\
&\quad + h \mathbf{f}_{\mathbf{y},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{y},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s v_i \\
\mu_n &= \mu_{n+1} + \sum_{i=1}^s \left((u_i \cdot \mathbf{f}_{\mathbf{p},\mathbf{y}}) \cdot k_i + (\hat{u}_i \cdot \mathbf{r}_{\mathbf{p},\mathbf{y}}) \cdot k_i \right) \\
&\quad + h \mathbf{f}_{\mathbf{p},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{p},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s \bar{v}_i \\
&= \mu_{n+1} + \sum_{i=1}^s \left((\mathbf{f}_{\mathbf{p},\mathbf{y}} \cdot k_i)^T \cdot u_i + \hat{u}_i (\mathbf{r}_{\mathbf{p},\mathbf{y}} \cdot k_i)^T \right) \\
&\quad + h \mathbf{f}_{\mathbf{p},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i u_i + h \mathbf{r}_{\mathbf{p},\mathbf{t}}^T(t_n, y_n, p) \cdot \sum_{i=1}^s \gamma_i \hat{u}_i + \sum_{i=1}^s \bar{v}_i.
\end{aligned}$$

The result is

$$(F.8a) \quad \tilde{\mathbf{R}}^T(\gamma, t_n, y_n) \cdot u_i = h b_i \mathbf{r}_{\mathbf{y}}^T + m_i \lambda_{n+1} + \sum_{j=i+1}^s \left(a_{j,i} v_j + \frac{c_{j,i}}{h} u_j \right)$$

$$(F.8b) \quad v_i = \mathbf{f}_{\mathbf{y}}^T(T_i, Y_i, p) \cdot u_i + h b_i \mathbf{r}_{\mathbf{y}}^T(T_i, Y_i, p)$$

$$(F.8c) \quad \bar{v}_i = \mathbf{f}_{\mathbf{p}}^T(T_i, Y_i, p) \cdot u_i + h b_i \mathbf{r}_{\mathbf{p}}^T(T_i, Y_i, p)$$

$$(F.8d) \quad \lambda_n = \lambda_{n+1} + \sum_{i=1}^s (\mathbf{f}_{\mathbf{y},\mathbf{y}} \cdot k_i)^T \cdot u_i + h \sum_{i=1}^s b_i (\mathbf{r}_{\mathbf{y},\mathbf{y}} \cdot k_i)^T \\ + h \mathbf{f}_{\mathbf{y},\mathbf{t}}^T \cdot \sum_{i=1}^s \gamma_i u_i + h^2 \rho \mathbf{r}_{\mathbf{y},\mathbf{t}}^T + \sum_{i=1}^s v_i$$

$$(F.8e) \quad \mu_n = \mu_{n+1} + \sum_{i=1}^s (\mathbf{f}_{\mathbf{p},\mathbf{y}} \cdot k_i)^T \cdot u_i + h \sum_{i=1}^s b_i (\mathbf{r}_{\mathbf{p},\mathbf{y}} \cdot k_i)^T \\ + h \mathbf{f}_{\mathbf{p},\mathbf{t}}^T \cdot \sum_{i=1}^s \gamma_i u_i + h^2 \rho \mathbf{r}_{\mathbf{p},\mathbf{t}}^T + \sum_{i=1}^s \bar{v}_i,$$

where $\rho = \sum_{i=1}^s \gamma_i b_i$ and all derivatives are evaluated at (t_n, y_n, p) , unless their arguments are explicitly shown.

When all partial derivatives with respect to parameters are set to zero in (F.8)

one obtains the discrete Rosenbrock adjoint (D.6), which calculates sensitivities with respect to initial conditions.

Supplementary material part G. FATODE error estimation and step size control.

Variable step size control is routinely adopted by general ODE solvers to control numerical errors and maximize efficiency. FATODE’s forward integrators use estimates of the truncation error to decide whether to accept or reject the step, and to compute the next step size. The maximum number of integration steps before an unsuccessful return can be specified by the user.

For the ERK, SDIRK, and Rosenbrock methods the classical error estimators based on embedded solutions are implemented; they proved to work well in practice. Two different error estimation options are provided for fully implicit Runge-Kutta methods. One is the classical error estimation [14] which uses an embedded third order method based on an additional explicit stage. The second estimator uses two additional stages: an explicit stage at the beginning of the time step and another SDIRK stage which re-uses the LU decomposition from the solution of the main integrator. The coefficients are chosen such that the order of consistency of the embedded solution \hat{y}_{n+1} is $\hat{p} = p - 1$, where p is the order of y_{n+1} . The difference vector $Est = \hat{y}_{n+1} - y_{n+1}$ is used as a local error estimator. Our experience indicates that the SDIRK error estimator yields better results overall.

The local error test is performed as follows. Let $Tol_k = atol_k + rtol_k \cdot |y_{n+1,k}|$, where $atol$ and $rtol$ are the absolute and relative error tolerance specified by user, and $|y_{n+1,k}|$ is the absolute value of the k -th component of y_{n+1} . The relative and absolute error tolerances can be either vector or scalar (in which case the same tolerance values are used for all components k). The local error test takes the form [14]

$$(G.1) \quad Err = \sqrt{\frac{1}{d} \sum_{k=1}^d \left(\frac{Est_k}{Tol_k} \right)^2} < 1.$$

If the test passes the step size is accepted, otherwise it rejected and the step is re-computed. The new step size, for both accepted and rejected cases, is given by [14]

$$h_{\text{new}} = h_{\text{old}} \cdot \min \left(\mathcal{F}_{\text{max}}, \max \left(\mathcal{F}_{\text{min}}, \mathcal{F}_{\text{safe}} \cdot Err^{-1/(\hat{p}+1)} \right) \right),$$

where \mathcal{F}_{max} is the upper bound on step increase factor, \mathcal{F}_{min} the lower bound on step decrease factor, and $\mathcal{F}_{\text{safe}}$ is a safety factor. The default values of these factors depend on the specific method. All of them can be changed by the user in the parameter settings. If the step size is rejected at the first step, the step increase factor \mathcal{F}_{max} is set to 1, and the step size is reduced by a factor of 10. Furthermore, the step size can be constrained by minimum (h_{min}) and maximum (h_{max}) values. The starting step size h_{start} can be specified by the user.

For the tangent linear model integration FATODE provides two options for controlling errors in the sensitivities. The first option is to use only the forward error estimates for step size control. The second option is to estimate the truncation errors for both the forward solution and the tangent linear model solution. The solution error is taken as the maximum between the forward truncation error and the truncation error of any column of the sensitivities. This solution error is then used to control the step size.

The discrete adjoint model integration traces the same sequence of steps as the forward integration, in reverse order. Therefore, the choice of the step sizes is completely determined during the forward integration, and the accuracy of the adjoint solution will depend on the error control performed during the forward run.

G.1. Computing derivatives required by FATODE. The implicit formulation of the stiff solvers, as well as the formulation of the tangent linear and adjoint methods, require the computation of various derivatives, summarized in Appendix A. These derivatives include the Jacobians of the ODE function with respect to the state \mathbf{f}_y , e.g., in equations (B.2); gradients of the quadrature function r_y^T, r_p^T , e.g., in (F.4); Jacobian times vector products, e.g., $\mathbf{f}_y u$ in (C.1); transposed Jacobians times vector products, e.g., $\mathbf{f}_y^T u$ and $\mathbf{f}_p^T u$ in (F.8b)–(F.8c); Hessian times vector products, e.g., $(\mathbf{f}_{y,yk}) u$ in (C.6b) and $(\mathbf{f}_{p,yk})^T u, (r_{p,yk})^T u$ in (F.8); and time derivatives of Jacobians transposed times vectors, e.g., $\mathbf{f}_{p,t}^T u$ in (F.8d). Among the methods implemented in FATODE, the Rosenbrock family requires the calculation of most derivatives, including Hessian vector products.

The derivatives supplied to FATODE can be obtained analytically, by finite differences, or by automatic differentiation. The errors in the derivative terms should be smaller than the local truncation error of the integrator, otherwise a loss of accuracy in the computed sensitivities will be experienced. Therefore utmost care must be exercised with the use of finite difference approximations. If analytical derivatives are not available, automatic differentiation tools like TAMC [11] can provide considerable help. For example, a Jacobian vector product is obtained by one TAMC run in forward mode, a transposed-Jacobian vector product by one TAMC run in reverse mode, and the product between the Hessian transposed times vector can be obtained by two consecutive runs of TAMC in forward mode.

Supplementary material part H. Linear solvers in FATODE. The most computationally intensive part in solving large-scale ODE systems by implicit methods is the solution of linear systems at each step. Linear systems arise from the simplified Newton iterations applied to solve the nonlinear systems in case of fully implicit Runge-Kutta methods and SDIRK methods. For Rosenbrock methods, linear systems appear directly in the formula (2.4). In general, all implicit time stepping methods in FATODE require the solution of linear systems with matrices \mathbf{R} or $\tilde{\mathbf{R}}$ defined in (B.2). These matrices inherit the sparsity structure of the system Jacobian.

The best efficiency is achieved when taking advantage of the problem-specific characteristics. Consequently, FATODE was designed to allow users to provide their own linear solvers and sparse data structures. We have incorporated three direct methods in current version of FATODE. For dense systems calls to LAPACK [3] routines are provided. For large sparse systems, substantial memory and execution time benefits can be gained by calling the direct sparse solvers UMFPACK [6] and SuperLU [7] and representing the sparse matrices in a compressed column format. Interfaces to both these codes are provided.

All the relevant information is encapsulated in a linear algebra module, as explained in Section 4. The module contains interfaces to the following four generic routines: *LS_Init* (initialization and memory allocation required by the specific linear solver), *LS-Decomp* (LU decomposition), *LS-Solve* (solves the triangular systems by substitution), and *LS-Free* (frees the memory allocated and clears the objects created during the initialization stage). The time integrators make calls to these functions without having to consider the underlying solver details. The user can choose one of the linear solvers provided (LAPACK, UMFPACK, SuperLU), or can use them as templates and add a new linear solver to the module. For example adding an iterative solver requires to provide a data structure for the Jacobian, and a corresponding Jacobian-vector product routine; *LS-Decomp* remains empty, while the iterative solver is called from within *LS-Solve*. All the required code modifications are within the linear algebra module.

Supplementary material part I. Numerical experiments with stiff solvers on the two dimensional shallow water problem.

I.1. Forward solution. In our test, we choose the option of the BDF method for the comparison with implicit methods in FATODE. From each family of implicit integrators in FATODE we select several representative methods with different orders for the tests: Lobatto3C, Radau2A, and Gauss (fully implicit Runge-Kutta); Ros3 and Ros4 (Rosenbrock); Sdirk4a, Sdirk2a (singly diagonally implicit Runge-Kutta). The Gustafsson predictive error controller is used for all integrators. An additional SDIRK stage was used in the error estimator in the fully implicit Runge-Kutta integrator.

Since solution of linear systems dominates the computational cost of implicit integration, especially for large-scale ODE systems, it is necessary to use the same linear solvers for both CVODE and FATODE for a fair performance comparison. In our comparison experiments we use the direct linear solver (DGETRF and DGETRS) from LAPACK simply for comparison purposes. Note that sparse linear solvers should be used in practice for best efficiency. In all cases, the full Jacobian is supplied.

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 simulation time interval is $[t_0, t_F] = [0, 0.02]$ time units. A reference solution 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 relative error is defined by (5.2).

The work-precision diagrams for the stiff solvers are shown in Figure I.1. The results indicate that singly diagonally implicit and fully implicit Runge-Kutta methods implemented in FATODE outperform the BDF method implemented in CVODE, requiring fewer time steps and considerably smaller CPU times to reach a desired accuracy. The Rosenbrock method is also more efficient than CVODE for accuracy levels below 10^{-6} .

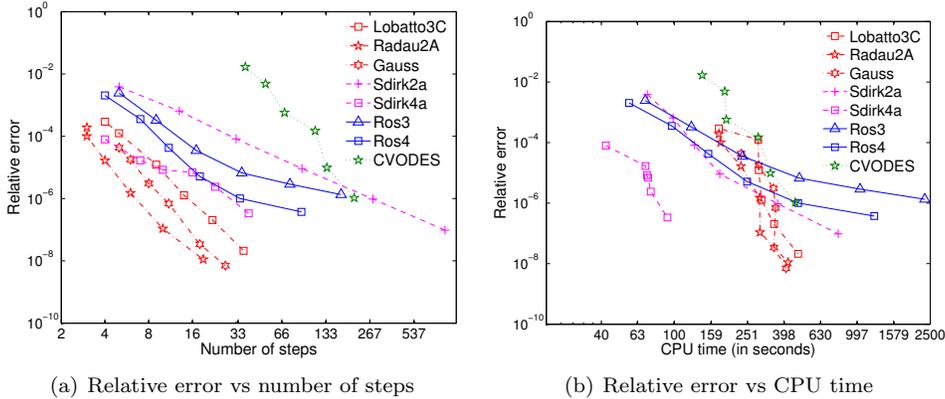


FIG. I.1. Forward integration of the shallow water equations (5.2) using stiff integrators. Comparison is made between FIRK, SDIRK, and Rosenbrock methods in FATODE with the BDF method in CVODE. Different points on the plots correspond to different absolute and relative tolerances levels in the range $10^{-2}, \dots, 10^{-7}$.

I.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 $\partial y_i(t_F)/\partial y_1(t_0)$, $i = 1 \dots n$ using tangent linear model integration. The

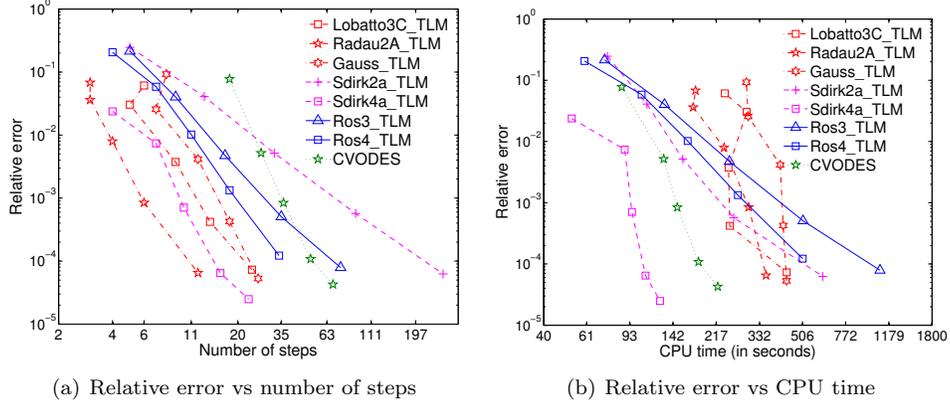


FIG. I.2. *Tangent linear model integration of the shallow water equations (5.2) using stiff solvers. Comparison is made between implicit methods in FATODE with the BDF method in CVODES. Different points on the plots correspond to different absolute and relative tolerances levels in the range $10^{-2}, \dots, 10^{-6}$. The tangent linear model computes the sensitivity $\partial y_i(t_F)/\partial y_1(t_0)$, $i = 1 \dots d$.*

FATODE results are compared against those obtained with CVODES [24], an extension of CVODE capable to perform sensitivity analysis. The LAPACK linear solvers are used in both CVODES and FATODE.

Tangent linear model results with the stiff integrators are shown in Figure I.2. The three implicit methods in FATODE requires considerably fewer steps than CVODES. The performance of the SDIRK and Rosenbrock methods is comparable to that of the BDF method in CVODES in terms of accuracy versus CPU time. The fully implicit Runge-Kutta method is nearly three times more expensive since it solves either a large real-valued system or a complex-valued system at each step.

I.3. Adjoint sensitivity analysis. We calculate the sensitivities of the first solution component at the final time with respect to all initial values $\partial y_1(t_F)/\partial y_i(t_0)$, $i = 1 \dots d$ using adjoint model integration. Work-precision diagrams for the implicit solvers are shown in Figure I.3. All implicit methods in FATODE outperform CVODES in terms of both number of steps and CPU time required for a given solution accuracy. The highest efficiency is achieved by the Rosenbrock method, despite the computation of Hessian-vector products.

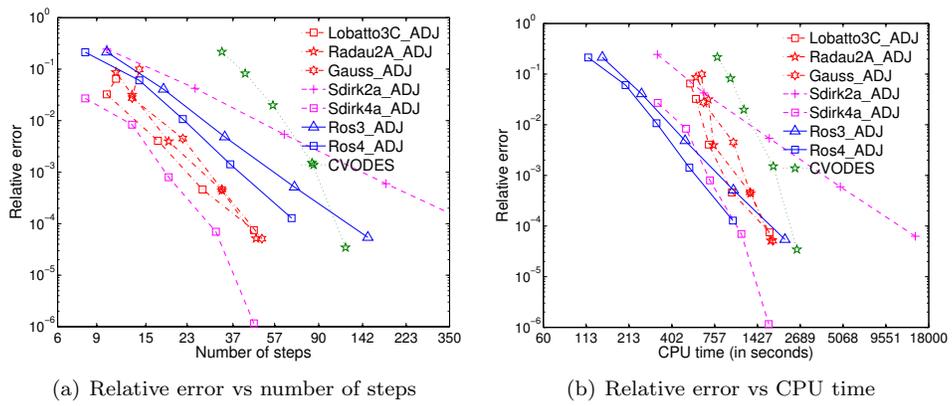


FIG. 1.3. Adjoint integration of the shallow water equations (5.2) using stiff solvers. Comparison is made between implicit methods in FATODE with the BDF method in CVODES. Different points on the plots correspond to different absolute and relative tolerances levels in the range $10^{-2}, \dots, 10^{-6}$. The adjoint model computes the sensitivity $\partial y_1(t_F)/\partial y_i(t_0)$, $i = 1 \dots d$.

Supplementary material part J. The Carbon Bond-IV (CBM-IV) chemical mechanism.

TABLE J.1
List of species in CBM-IV

No.	Species	Initial concentrations (in molecules/cm ³)	Absolute tolerances (<i>ATOL</i> ₀)
1	OID	3.65E-2	1E-9
2	H2O2	3.47E11	1E4
3	PAN	2.56E3	1E-4
4	CRO	3.35E-23	1E-30
5	TOL	5.29E-20	1E-27
6	N2O5	1.31E7	1
7	XYL	0	1E-14
8	XO2N	1.99E1	1E-6
9	HONO	3.84E8	1E1
10	PNA	2.68E8	1E1
11	TO2	1.59E-24	1E-31
12	HNO3	1.19E12	1E5
13	ROR	4.89E-5	1E-12
14	CRES	5.46E-21	1E-28
15	MGLY	1.94E-23	1E-30
16	CO	2.33E12	1E5
17	ETH	2.11E-30	1E-37
18	XO2	5.51E8	1E1
19	OPEN	2.63E-21	1E-28
20	PAR	8.12E3	1E-4
21	HCHO	3.00E10	1E3
22	ISOP	0	1E-14
23	OLE	0	1E-14
24	ALD2	2.69E2	1E-5
25	O3	2.06E12	1E5
26	NO2	1.56E10	1E3
27	OH	3.67E7	1
28	HO2	9.89E8	1E1
29	O	1.25E4	1E-3
30	NO3	3.36E7	1
31	NO	2.73E9	1E2
32	C2O3	6.53	1E-7

TABLE J.2
The first 6 dominant eigenvalues of Jacobian matrix in CBM-IV

Rank	Eigenvalues
1	-1.404 53E9
2	-7.219 91E4
3	-3.749 62E3
4	-4.222 09
5	-2.272 99
6	-2.442 18E-1

TABLE J.3
List of reactions in CBM-IV

No.	Reactions	No.	Reactions
1	NO ₂ +hν=NO+O	42	ALD2+O=C2O3+OH
2	O+O ₂ +M=O ₃	43	ALD2+OH=C2O3
3	O ₃ +NO=NO ₂	44	ALD2+NO ₃ =C2O3+HNO ₃
4	O+NO ₂ =NO	45	ALD2+hν+2O ₂ =HCHO+XO ₂ +CO+2HO ₂
5	O+NO ₂ =NO ₃	46	C2O3+NO=HCHO+XO ₂ +HO ₂ +NO ₂
6	O+NO=NO ₂	47	C2O3+NO ₂ =PAN
7	O ₃ +NO ₂ =NO ₃	48	PAN=C2O3+NO ₂
8	O ₃ +hν=O	49	2C2O3=2HCHO+2XO ₂ +2HO ₂
9	O ₃ +hν=O1D	50	C2O3+HO ₂ =0.79HCHO+0.79XO ₂ +0.79HO ₂ +0.79OH
10	O1D=O	51	OH=HCHO+XO ₂ +HO ₂
11	O1D+H ₂ O=2OH	52	PAR+OH=0.87XO ₂ +0.13XO ₂ N+0.11HO ₂ +0.11ALD2 +0.76ROR-0.11PAR
12	O ₃ +OH=HO ₂	53	ROR=1.1ALD2+0.96XO ₂ +0.94HO ₂ +0.04XO ₂ N +0.02ROR-2.10PAR
13	O ₃ +HO ₂ =OH	54	ROR=HO ₂
14	NO ₃ +hν=0.89NO ₂ +0.89O+0.11NO	55	ROR+NO ₂ =PROD
15	NO ₃ +NO=2NO ₂	56	O+OLE=0.63ALD2+0.38HO ₂ +0.28XO ₂ +0.3CO +0.2 HCHO+0.02XO ₂ N+0.22PAR+0.2OH
16	NO ₃ +NO ₂ =NO+NO ₂	57	OH+OLE=HCHO+ALD2+XO ₂ +HO ₂ -PAR
17	NO ₃ +NO ₂ =N ₂ O ₅	58	O ₃ +OLE=0.5ALD2+0.74HCHO+0.33CO+0.44HO ₂ + 0.22XO ₂ +0.1OH-PAR
18	N ₂ O ₅ +H ₂ O=2HNO ₃	59	NO ₃ +OLE=0.91XO ₂ +HCHO+ALD2+0.09XO ₂ N +NO ₂ -PAR
19	N ₂ O ₅ =NO ₃ +NO ₂	60	O+ETH=HCHO+0.7XO ₂ +CO+1.7HO ₂ +0.3OH
20	2NO=2NO ₂	61	OH+ETH=XO ₂ +1.56HCHO+HO ₂ +0.22ALD2
21	NO+NO ₂ +H ₂ O=2HONO	62	O ₃ +ETH=HCHO+0.42CO+0.12HO ₂
22	OH+NO=HONO	63	OH+TOL=0.08XO ₂ +0.36CRES+0.44HO ₂ +0.56TO ₂
23	HONO+hν=OH+NO	64	TO ₂ +NO=0.9NO ₂ +0.9OPEN+0.9HO ₂
24	OH+HONO=NO ₂	65	TO ₂ =HO ₂ +CRES
25	2HONO=NO+NO ₂	66	OH+CRES=0.4CRO+0.6XO ₂ +0.6HO ₂ +0.3OPEN
26	OH+NO ₂ =HNO ₃	67	NO ₃ +CRES=CRO+HNO ₃
27	OH+HNO ₃ =NO ₃	68	CRO+NO ₂ =PROD
28	HO ₂ +NO=OH+NO ₂	69	OH+XYL=0.7HO ₂ +0.5XO ₂ +0.2CRES+0.8MGLY +1.10 PAR+0.3TO ₂
29	HO ₂ +NO ₂ =PNA	70	OH+OPEN=XO ₂ +C2O3+2HO ₂ +2CO+HCHO
30	PNA=HO ₂ +NO ₂	71	OPEN+hν=C2O3+CO+HO ₂
31	OH+PNA=NO ₂	72	O ₃ +OPEN=0.03ALD2+0.62C2O3+0.7HCHO+0.03XO ₂ +0.69CO+0.08OH+0.76HO ₂ +0.2MGLY
32	2HO ₂ =H ₂ O ₂	73	OH+MGLY=XO ₂ +C2O3
33	2HO ₂ +H ₂ O=H ₂ O ₂	74	MGLY+hν=C2O3+CO+HO ₂
34	H ₂ O ₂ +hν=2OH	75	O+ISOP=0.6HO ₂ +0.8ALD2+0.55OLE+0.5XO ₂ +0.5CO+0.45ETH+0.9PAR
35	OH+H ₂ O ₂ =HO ₂	76	OH+ISOP=HCHO+XO ₂ +0.67HO ₂ +0.4MGLY +0.2C2O3+ETH+0.2ALD2+0.13XO ₂ N
36	OH+CO=HO ₂	77	O ₃ +ISOP=HCHO+0.4ALD2+0.55ETH+0.2MGLY +0.06CO+ 0.1PAR+0.44HO ₂ +0.1OH
37	HCHO+OH=HO ₂ +CO	78	NO ₃ +ISOP=XO ₂ N
38	HCHO+hν+2O ₂ =2HO ₂ +CO	79	XO ₂ +NO=NO ₂
39	HCHO+hν=CO	80	2XO ₂ =PROD
40	HCHO+O=OH+HO ₂ +CO	81	XO ₂ N+NO=PROD
41	HCHO+NO ₃ =HNO ₃ +HO ₂ +CO		

TABLE J.4
List of reaction rates in CBM-IV

No.	Reaction rate	No.	Reaction rate
1	(8.89E-3)*SUN	42	ARR2(1.2E-11, -9.86E2)
2	ARR2(1.4E3, 1.175E3)	43	ARR2(7.0E-12, 2.5E2)
3	ARR2(1.8E-12, -1.37E3)	44	2.5E-15
4	9.3E-12	45	(4.0E-6)*SUN
5	ARR2(1.6E-13, 6.87E2)	46	ARR2(5.4E-12, 2.5E2)
6	ARR2(2.2E-13, 6.02E2)	47	ARR2(8.0E-20, 5.5E3)
7	ARR2(1.2E-13, -2.45E3)	48	ARR2(9.4E16, -1.4E4)
8	(3.556E-4)*SUN	49	2.0E-12
9	(2.489E-5)*SUN	50	6.5E-12
10	ARR2(1.9E8, 3.9E2)	51	ARR2(1.1E2, -1.71E3)
11	2.2E-10	52	8.1E-13
12	ARR2(1.6E-12, -9.4E2)	53	ARR2(1.0E15, -8.0E3)
13	ARR2(1.4E-14, -5.8E2)	54	1.6E3
14	(1.378E-1)*SUN	55	1.5E-11
15	ARR2(1.3E-11, 2.5E2)	56	ARR2(1.2E-11, -3.24E2)
16	ARR2(2.5E-14, -1.23E3)	57	ARR2(5.2E-12, 5.04E2)
17	ARR2(5.3E-13, 2.56E2)	58	ARR2(1.4E-14, -2.105E3)
18	1.3E-21	59	7.7E-15
19	ARR2(3.5E14, -1.0897E4)	60	ARR2(1.0E-11, -7.92E2)
20	ARR2(1.8E-20, 5.3E2)	61	ARR2(2.0E-12, 4.11E2)
21	4.39999E-40	62	ARR2(1.3E-14, -2.633E3)
22	ARR2(4.5E-13, 8.06E2)	63	ARR2(2.1E-12, 3.22E2)
23	(1.511E-3)*SUN	64	8.1E-12
24	6.6E-12	65	4.2
25	1.0E-20	66	4.1E-11
26	ARR2(1.0E-12, 7.13E2)	67	2.2E-11
27	ARR2(5.1E-15, 1.0E3)	68	1.4E-11
28	ARR2(3.7E-12, 2.40E2)	69	ARR2(1.7E-11, 1.16E2)
29	ARR2(1.2E-13, 7.49E2)	70	3.0E-11
30	ARR2(4.8E13, -1.0121E4)	71	(5.334E-5)*SUN
31	ARR2(1.3E-12, 3.8E2)	72	ARR2(5.4E-17, -5.0E2)
32	ARR2(5.9E-14, 1.15E3)	73	1.7E-11
33	ARR2(2.2E-38, 5.8E3)	74	(1.654E-4)*SUN
34	(6.312E-6)*SUN	75	1.8E-11
35	ARR2(3.1E-12, -1.87E2)	76	9.6E-11
36	2.2E-13	77	1.2E-17
37	1.0E-11	78	3.2E-13
38	(2.845E-5)*SUN	79	8.1E-12
39	(3.734E-5)*SUN	80	ARR2(1.7E-14, 1.3E3)
40	ARR2(3.0E-11, -1.55E3)	81	6.8E-13
41	6.3E-16		

1. The coefficient SUN is updated with time. It is 0 during the night. During daytime, it is computed as $(1 + \cos(\pi * ((2 * tl - tsr - tss)/(tss - tsr))^2))/2$ where tl is the local time, tsr and tss correspond to the sunrise time and sunset time respectively. In our experiments, we use $tsr = 4.5$ (4:30am) and $tss = 19.5$ (7:30pm).
2. Function $ARR2(a, b)$ is defined as $a * \exp(b/temperature)$.

Supplementary material part K. Numerical experiments with stiff solvers on the CBM-IV problem.

TABLE K.1
The sensitivities of five species to reaction rates at a temperature of 298K

Reaction No.	N_2O_5	$HONO$	HNO_3	O_3	NO_2
4	-8.28E13	2.36E15	-2.21E16	-1.44E19	-2.10E14
11	1.39E16	5.36E17	-1.72E18	-1.27E21	3.03E17
18	-1.21E27	-4.39E27	2.10E29	-1.61E31	-1.85E29
21	2.14E37	1.52E41	-1.50E41	-1.15E42	-3.57E39
24	-3.28E15	-8.91E18	8.89E18	1.64E20	2.78E17
25	-4.94E16	-1.79E20	1.70E20	-2.62E21	6.06E18
36	-6.93E18	-1.11E20	-1.37E20	-2.57E23	-1.03E20
37	-4.24E16	-4.69E17	-1.79E18	-1.73E21	-4.20E17
41	-1.88E19	2.88E20	-5.84E20	-2.93E24	-2.43E21
44	3.67E9	-6.93E10	2.35E12	6.27E14	-2.94E11
49	-8.63E2	2.03E4	-1.85E5	-1.65E8	2.35E3
50	-2.18E8	5.25E9	2.64E11	-4.40E13	-4.22E10
52	-6.94E8	9.39E10	1.21E13	-3.66E14	-3.30E11
54	-1.43E-6	-1.45E-5	-5.27E-2	-7.29E-2	-6.78E-4
55	-3.65E5	-3.03E6	-1.26E10	-1.95E10	-1.66E8
64	2.53E-16	4.80E-16	9.13E-12	1.95E-11	1.17E-13
65	-4.88E-28	-9.26E-28	-1.76E-23	-3.76E-23	-2.26E-25
66	5.67E-15	1.39E-14	1.74E-10	4.38E-10	2.24E-12
67	-1.05E-14	-2.54E-14	-3.22E-10	-8.15E-10	-4.15E-12
68	-5.72E-20	-1.94E-19	-5.36E-18	-4.71E-15	-1.93E-19
70	-3.25E-17	-9.29E-16	2.15E-15	3.86E-13	-8.02E-16
74	6.77E-19	9.25E-18	3.11E-17	3.12E-14	7.26E-18
79	4.82E16	-1.73E18	1.53E19	8.36E21	-1.37E17
81	3.86E6	2.99E8	-4.22E10	-3.63E11	6.04E9

Supplementary material part L. Results with different approaches to solve the adjoint system within each step.

TABLE L.1

Timings and accuracy of the CBM-IV test and the shallow water test solving the linear adjoint system directly (or via simplified Newton iterations)

Method	Tolerances		Relative error	CPU time
	Rtol	Atol		
CBM-IV				
Sdirk4a	1E-3	$atol_0$	4.307E-6 (1.933E-5)	52.25ms (93.43ms)
	2E-4	$0.2 \times atol_0$	5.757E-6 (1.231E-5)	77.41ms (14.73ms)
Radau2A	1E-3	$atol_0$	3.047E-6 (2.997E-6)	55.95ms (69.99ms)
	2E-4	$0.2 \times atol_0$	1.454E-6 (1.436E-6)	61.04ms (73.25ms)
Shallow water				
Sdirk4a	1E-2	1E-2	2.697E-2 (2.697E-2)	14.41s (11.68s)
	1E-3	1E-3	8.349E-2 (8.349E-2)	20.22s (17.37s)
Radau2A	1E-2	1E-2	8.669E-2 (8.669E-2)	160.06s (32.85s)
	1E-3	1E-3	3.170E-2 (3.170E-2)	203.75s (39.82s)

Acknowledgements. 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.