Forward, Tangent Linear, and Adjoint Runge Kutta Methods in KPP–2.2 for Efficient Chemical Kinetic Simulations

Adrian Sandu* and Philipp Miehe

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, VA 24061.

Email: {sandu, pmiehe}@cs.vt.edu

July 7, 2006

*Corresponding author. Phone: (540)-231-2193. Fax: (540)-231-6075. Email: sandu@cs.vt.edu.

Abstract

The Kinetic PreProcessor (KPP) is a widely used software environment which generates Fortran90, Fortran77, Matlab, or C code for the simulation of chemical kinetic systems. High computational efficiency is attained by exploiting the sparsity pattern of the Jacobian and Hessian. In this paper we report on the implementation of two new families of stiff numerical integrators in the new version 2.2 of KPP. One family is the fully implicit three-stage Runge Kutta methods, and the second family are singly diagonally-implicit Runge Kutta methods. For each family tangent linear models for direct decoupled sensitivity analysis, and adjoint models for adjoint sensitivity analysis of chemical kinetic systems are also implemented. To the best of our knowledge this work brings the first implementation of the direct decoupled sensitivity method and of the discrete adjoint sensitivity method with Runge Kutta methods. Numerical experiments with a chemical system used in atmospheric chemistry illustrate the power of the stiff Runge Kutta integrators and their tangent linear and discrete adjoint models. Through the integration with KPP–2.2. these numerical techniques become easily available to a wide community interested in the simulation of chemical kinetic systems.

Keywords: Stiff chemical kinetics, Runge Kutta methods, direct decoupled sensitivity analysis, adjoint sensitivity analysis

1 Introduction

Computer simulation of chemical kinetic systems requires efficient implementations of reaction mechanisms for analysis and further development. Numerical stiffness, due to the presence of both fast and slow reactions with reaction speeds spanning many orders of magnitude, poses special challenges to the numerical integration techniques. A number of numerical codes have been developed to integrate stiff ordinary differential equations (ODEs) describing chemical kinetics, e.g., Facsimile [12], AutoChem [1], SPACK [16], CHEMKIN [2], ODEPACK [7], and KPP [14].

The Kinetic PreProcessor KPP [14, 26, 13, 27] is a software tool that assists the computer simulation of chemical kinetic systems. The concentrations of a chemical system evolve in time according to the differential law of mass action kinetics. A numerical simulation requires an implementation of the differential laws and a numerical integration in time. KPP is currently being used by many academic, research, and industry groups in several countries [5, 33, 34, 32, 31, 24].

KPP translates a specification of the chemical mechanism into Fortran77, Fortran90, C, or Matlab simulation code that implements the concentration time derivative function, its Jacobian, and it Hessian, together with a suitable numerical integration scheme. Sparsity in Jacobian/Hessian is carefully exploited in order to obtain computational efficiency. Fortran90 is the programming language of choice for the vast majority of scientific applications. Matlab [6] provides a high-level programming environment for algorithm development, numerical computations, and data analysis and visualization. The Matlab code produced by KPP allows a rapid implementation and analysis of a specific chemical mechanism.

A summary of KPP generated routines is given below:

1. *Fun*: the time derivative of concentrations;
2. *Jac, Jac_SP*: Jacobian of Fun in full or in sparse format;
3. *KppDecomp*: sparse LU decomposition for the Jacobian;
4. *KppSolve, KppSolveTR*: solve sparse system with the Jacobian matrix and its transpose;
5. *Jac_SP_Vec, JacTR_SP_Vec*: sparse Jacobian (transposed or not) times vector;

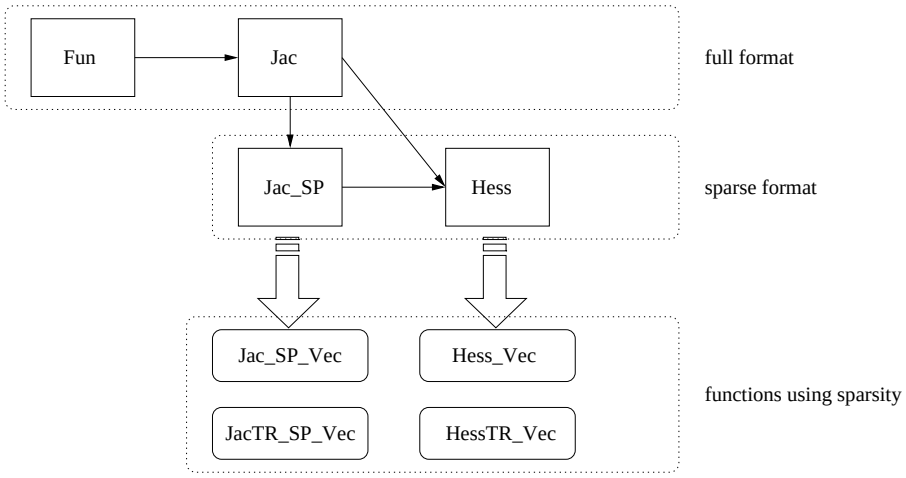

Figure 1: KPP-generated building blocks for chemistry simulations include the derivative function, the Jacobian and the Hessian, sparsity information on the Jacobian and the Hessian, and sparse linear algebra routines for tensor-vector products and the solution of sparse linear systems.

6. The stoichiometric matrix STOICM;

7. *ReactantProd*: vector of reaction rates;

8. *JacReactantProd*: the Jacobian of the above;

9. *dFun_dRcoeff*: derivatives of Fun with respect to reaction coefficients (in sparse format);

10. *dJac_dRcoeff*: derivatives of Jac with respect to reaction coefficients times user vector;

11. *Hess*: the Hessian of Fun; this 3-tensor is represented in sparse format;

12. *Hess_Vec, HessTR_Vec*: Hessian (or its transpose) times user vectors; same as the derivative of Jacobian (transposed) vector product times vector.

Figure 1 summarizes the main components of the KPP-generated code that are useful for most chemistry simulations. Efficiency is provided through the sparsity format of the building blocks: only non-zero entries of the Jacobian as well as the Hessian are stored. Sparsity structure then requires functions to multiply the Jacobian and Hessian with a vector. The names of these functions are given in the figure.

The KPP environment allows for rapid prototyping of new chemical kinetic schemes as well as new numerical integration methods. KPP incorporates a library with several widely used atmospheric chem-

istry mechanisms; the users can add their own chemical mechanisms to the library. KPP also includes a comprehensive suite of stiff numerical integrators.

The original set of integrators in KPP–2.1 [14, 26, 13, 27] contains several implementations of efficient solvers employing the sparsity structure of the stiff system. A number of Rosenbrock methods of various orders can already be used in KPP [28]. The high efficiency has made Rosenbrock methods one of the first choices for many applications, and in particular for atmospheric chemistry. SEULEX [21], a variable order stiff extrapolation implementation, produces extremely accurate solution. Backward differentiation formula (BDF) methods have been used in the Livermore ODE solver (LSODE, LSODES [23]) in the implementation of the VODE [10] integrator to be applied to stiff problems. Furthermore, a BDF-based direct-decoupled sensitivity integrator is part of the original set of solvers by modifying ODESSA [22] to use the sparse linear algebra routines of KPP.

In this paper we describe two new families of stiff numerical integrators which have been implemented in the new version 2.2 of KPP. One family is the fully implicit three-stage Runge Kutta methods (including Radau-1A, Radau-2A, Lobatto-3C and Gauss formulas). The second family are singly diagonally-implicit Runge Kutta (SDIRK) methods (including five different formulas of orders 2–4). Integrators in both these families have excellent stability properties and allow for efficient and high accuracy solutions of chemical kinetic systems. In addition, tangent linear models for direct decoupled sensitivity analysis, and adjoint models for adjoint sensitivity analysis of chemical kinetic systems are also implemented. The implementation of the new integrators is done in Fortran90.

The paper brings the following novel elements: (1) this is the first implementation of multiple Runge Kutta methods from both fully implicit and SDIRK families; (2) to the best of our knowledge this is the first implementation of the direct decoupled method for sensitivity analysis with Runge Kutta methods; (3) to the best of our knowledge this is the first implementation of discrete adjoint implicit Runge Kutta methods.

The paper is organized as follows. A short overview of KPP is given. The new integrators are presented in section 2. Mathematical background is presented here and implementation details are explained. Section 6 shows results from applying the solvers to a chemical system. Comparisons of the

different implementation details are elaborated in detail. In section 7 a data assimilation example using our integrators and sensitivity analysis based on our implementation is briefly presented. Finally, the conclusion is drawn in section 9.

2 The Runge Kutta Numerical Integrators

In KPP version 2.2, stiff numerical integrators of the Runge Kutta family have been added to the KPP numerical library of stiff solvers for chemical kinetic system ODEs.

Consider a chemical kinetic system defined by the stiff system of ordinary differential equations

$$y' = f\left(t, y\right) \ , \quad t^0 \leq t \leq t^{\mathrm{F}} \ , \quad y\left(t^0\right) = y^0 \ , \quad y(t) \in \Re^n \ . \tag{1}$$

There are n chemical species in the system. The right hand side function $f(t, y) \in \Re^n$ describes the rates of change due to chemical production and loss processes. We will denote the Jacobian of the ODE function by $J(t, y) = \partial f / \partial y \in \Re^{n \times n}$. Typically the ODE system (1) is stiff: different chemical species change at very different rates during the kinetic evolution, and one is interested to follow the system evolution at the slower time scales. As a consequence the eigenvalues of the Jacobian differ by orders of magnitude. The numerical methods used to solve (1) have to be stable in the presence of stiffness [21]. In this paper we focus on implicit Runge Kutta integrators which have the necessary stability properties. We discuss in detail the implementation of stiff numerical Runge Kutta integrators in the numeric library of KPP–2.2.

A general s-stage Runge Kutta method is defined as [20]

$$\begin{aligned} y^{n+1} &= y^n + h \sum_{i=1}^{s} b_i k_i \ , \\ T_i &= t^n + c_i h \ , \quad Y_i = y^n + h \sum_{j=1}^{s} a_{ij} k_j \ , \\ k_i &= f\left(T_i, Y_i\right) \ . \end{aligned} \tag{2}$$

The coefficients a_{ij}, b_i and c_i determine the particular method and its accuracy and stability properties.

A particular Runge Kutta method can be compactly represented by the Butcher tableau of its coefficients:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \quad \equiv \quad \begin{array}{c|c} c & A \\ \hline & b^T \end{array} \tag{3}$$

The linear stability analysis considers the solution of the method (2) when applied to a linear scalar equation $y' = \lambda y$:

$$y^{n+1} = R(h\lambda)\, y^n \ , \quad R(z) = 1 + z\, b^T \, (I_s - zA)^{-1} \ \mathbb{1}_s \ , \quad \mathbb{1}_s = [1, \cdots, 1]^T \ .$$

When the real part $Re(\lambda) < 0$ the exact solution decreases to zero, and so should the numerical solution; this happens if the magnitude of the transfer function is $|R(z)| \leq 1$. The method (2) is A-stable if $|R(z)| \leq 1$ for all complex arguments with $Re(z) \leq 0$. The method is L-stable if, in addition to being A-stable, the transfer function satisfies $Re(\infty) = 0$ (complete damping of high frequencies). A Runge Kutta method is called *stiffly accurate* if the last stage solution and the final solution coincide; i.e. if $a_{sj} = b_j$ for $j = 1, \cdots, s$. Stiff accuracy is an essential property in the solution of differential algebraic equations [21] and is useful in the solution of stiff ordinary differential equations. All the Runge Kutta methods implemented in KPP–2.2 are A-stable, some are L-stable, and some are stiffly accurate. These properties will be highlighted when the methods are discussed.

2.1 Implementation Aspects

Following [21, Section IV.8], for implementation purposes the method (2) is written in terms of the variables $Z_i = Y_i - y^n$ as follows:

$$\begin{aligned} y^{n+1} &= y^n + h \sum_{i=1}^{s} b_i \, f\,(\, T_i,\, y^n + Z_i\,) = y^n + \sum_{i=1}^{s} d_i \, Z_i \ , \\ T_i &= t^n + c_i h \ , \quad Z_i = h \sum_{j=1}^{s} a_{ij} \, f\,(\, T_j,\, y^n + Z_j\,) \ . \end{aligned} \tag{4}$$

The stage relations

$$\begin{aligned} Z_1 &= h \sum_{j=1}^{s} a_{1j} f(T_j, y^n + Z_j) , \\ Z_2 &= h \sum_{j=1}^{s} a_{2j} f(T_j, y^n + Z_j) , \\ &\vdots \\ Z_s &= h \sum_{j=1}^{s} a_{sj} f(T_j, y^n + Z_j) \end{aligned} \tag{5}$$

form a nonlinear system of dimension $ns \times ns$ in the variables $Z_1 \cdots Z_s$ which needs to be solved at each time step. Replacing the nonlinear system in k_i in (2) by a nonlinear system in Z_i in (4) has numerical advantages for stiff systems where f has a large Lipschitz constant.

Explicit Runge Kutta methods. If the method coefficients are chosen such that $a_{ij} = 0$ for $j \geq i$ then the Runge Kutta method is explicit and no solutions of nonlinear systems are necessary. The stage vectors are obtained one after another by successive substitutions from the first stage to the last

$$\begin{aligned} Z_1 &= 0 , \\ Z_2 &= h\, a_{21} f(t^n, y^n) , \\ Z_3 &= h\, a_{31} f(t^n, y^n) + h\, a_{32} f(T_2, y^n + Z_2) , \\ &\vdots \\ Z_s &= h \sum_{j=1}^{s-1} a_{sj} f(T_j, y^n + Z_j) . \end{aligned}$$

While computationally inexpensive, explicit methods cannot offer the stability properties needed to solve stiff chemical equations. They are not considered for implementation in KPP–2.2.

Singly Diagonally-Implicit Runge Kutta (SDIRK) methods. Singly diagonally-implicit Runge Kutta (SDIRK) methods are characterized by coefficients satisfying $a_{ij} = 0$ for $j > i$ and $a_{ii} = \gamma$ for all i. Each stage leads to a nonlinear system of dimension $n \times n$; the stage vectors are obtained one after

another by solving the nonlinear systems for each stage in succession

$$\begin{aligned} Z_1 &= h\,\gamma\, f\,(\,T_1,\, y^n + Z_1\,)\ , \\ Z_2 &= h\,a_{21}\, f\,(\,T_1,\, y^n + Z_1\,) + h\,\gamma\, f\,(\,T_2,\, y^n + Z_2\,)\ , \\ &\vdots \\ Z_s &= h \sum_{j=1}^{s-1} a_{sj}\, f\,(\,T_j,\, y^n + Z_j\,) + h\,\gamma\, f\,(\,T_s,\, y^n + Z_s\,)\ . \end{aligned}$$

The nonlinear system that defines a stage vector Z_i

$$Z_i = h \sum_{j=1}^{i-1} a_{ij}\, f\,(\,T_j,\, y^n + Z_j\,) + h\,\gamma\, f\,(\,T_i,\, y^n + Z_i\,)$$

is solved by simplified Newton iterations of the form

$$\begin{aligned} &\Big[I_n - h\,\gamma\, J\,(\,t^n,\, y^n\,)\Big] \cdot \Delta Z_i^{[m]} = Z_i^{[m]} - h \sum_{j=1}^{i-1} a_{ij}\, f\,(\,T_j,\, y^n + Z_j\,) \\ &Z_i^{[m+1]} = Z_i^{[m]} - \Delta Z_i^{[m]}\ , \qquad m = 0, 1, \cdots \end{aligned} \tag{6}$$

The starting value for the Newton iterations $Z_i^{[0]}$ are chosen as zero, or are estimated by interpolation from the previously computed stage vectors $Z_1, \cdots, Z_{i-1}$. For all stages $i = 1, \cdots, s$ and for all iterations m the linear systems solved share the same matrix $I_n - h\,\gamma\, J$. Consequently the expensive LU decomposition of this $n \times n$ matrix is computed only once per time step.

Fully Implicit Runge Kutta methods. In the general case the method array of coefficients $A = (a_{ij})$ does not have a triangular structure, and the nonlinear system (5) of dimension $ns \times ns$ cannot be decoupled in a sequence of smaller systems. With the compact notation

$$Z = \Big[Z_1 \cdots Z_s\Big]^T\ , \quad F(Z) = \Big[f\,(\,T_1,\, y^n + Z_1\,) \cdots f\,(\,T_s,\, y^n + Z_s)\Big]^T\ ,$$

the nonlinear system (5) in Z can be written as [21]

$$Z = (hA \otimes I_n) \cdot F(Z)\ , \tag{7}$$

where $\otimes$ denotes the Kronecker product. The Kronecker product of two matrices $P = (p_{ij}) \in \mathbb{R}^{n\times n}$ and $Q = (q_{ij}) \in \mathbb{R}^{m\times m}$ is defined as

$$P \otimes Q = \begin{bmatrix} p_{11}Q & p_{12}Q & \dots & p_{1n}Q \\ p_{21}Q & p_{22}Q & \dots & p_{2n}Q \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1}Q & p_{n2}Q & \dots & p_{nn}Q \end{bmatrix} . \tag{8}$$

Following [21] the system (7) is solved by simplified Newton iterations of the form

$$\begin{aligned} &\Big[I_{ns} - hA \otimes J\,(\,t^n,\, y^n\,)\Big] \cdot \Delta Z^{[m]} = Z^{[m]} - (hA \otimes I_n)\; F\big(Z^{[m]}\big) \\ &Z^{[m+1]} = Z^{[m]} - \Delta Z^{[m]}\;, \quad m = 0, 1, \cdots \end{aligned} \tag{9}$$

Note that the system matrix is constructed using only the chemical Jacobian value at the beginning of the time step $J(t^n, y^n)$. The linear systems in (9) have dimension $ns \times ns$. Following [21] the KPP–2.2 implementation uses a transformation of the system (9) to complex form. This allows to replace the costly ns-dimensional real LU decomposition by several n-dimensional LU decompositions of real and complex matrices.

Differences regarding order and accuracy of the implicit Runge Kutta and SDIRK methods are discussed next.

2.2 Fully implicit Runge Kutta methods in KPP–2.2

The KPP–2.2 contains implementations of four different fully implicit Runge Kutta methods as follows:

- Radau-2A: stiffly accurate three stage method of order 5, based on the Radau-IIA quadrature. It is one of the most robust formulas for stiff ordinary differential equations [21].
- Lobatto-3C: stiffly accurate, three stage method of order 4 based on Lobatto quadrature.
- Gauss: three stage method of order 6 based on Gaussian quadrature. The method is only weakly L-stable ($R(\infty) = -1$) and is not stiffly accurate.
- Radau-1A: L-stable, three-stage method of order 5, based on Radau-IA quadrature. It is not stiffly accurate.

All methods are A-stable. Each of them has 3 stages, and require one complex and one real LU decomposition at each time step. The implementation has been inspired by the Radau5 code of Hairer and Wanner [21]. The Butcher tableaux of coefficients are presented in table 1. More information on these methods can be found in [21].

The selection of the fully-implicit Runge Kutta family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

```
#INTEGRATOR Runge_Kutta
```

The selection of a particular method within this family is done via the input control vector entry ICNTRL(3) (the values 0 or 1 select Radau-2A, 2 selects Lobatto-3C, 3 selects Gauss, and 4 selects Radau-1A).

2.3 SDIRK methods in KPP–2.2

KPP–2.2 contains implementations of five different SDIRK methods as follows:

- Sdirk-2 are two-stage, L-stable, stiffly accurate methods of order 2. The choice $\gamma = 1 - \sqrt{2}/2$ (Sdirk-2a) is more accurate, while the choice $\gamma = 1 + \sqrt{2}/2$ (Sdirk-2b) may be advantageous when a non-negative numerical solution (concentrations) is needed.

- Sdirk-3a is a three-stage, second order, stiffly accurate method. Its coefficients are chosen such that the discrete adjoint is also stiffly accurate.

- The methods Sdirk-4 are the fourth order L-stable singly-diagonally-implicit Runge Kutta methods developed by Hairer and Wanner [21]. Specifically, Sdirk-4a is the method with $\gamma = 4/15$ and Sdirk-4b the method with $\gamma = 1/4$. The coefficients of these methods are given in [21] and not reproduced here.

All methods are A-stable. Each of them requires a single real LU decomposition at each time step. The implementation has been inspired by the Sdirk4 code of Hairer and Wanner [21]. The Butcher tableaux of coefficients are presented in table 2.

The selection of the singly diagonally-implicit Runge Kutta family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

(a) Radau-2A (order 5, stiffly accurate)

0	$\frac{1}{6}$	$-\frac{1}{3}$	$\frac{1}{6}$
$\frac{1}{2}$	$\frac{1}{6}$	$\frac{5}{12}$	$-\frac{1}{12}$
1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

(b) Lobatto-3C (order 4, stiffly accurate)

$\frac{1}{2}-\frac{\sqrt{15}}{10}$	$\frac{5}{36}$	$\frac{2}{9}-\frac{\sqrt{15}}{15}$	$\frac{5}{36}-\frac{\sqrt{15}}{30}$
$\frac{1}{2}$	$\frac{5}{36}+\frac{\sqrt{15}}{24}$	$\frac{2}{9}$	$\frac{5}{36}-\frac{\sqrt{15}}{24}$
$\frac{1}{2}+\frac{\sqrt{15}}{10}$	$\frac{5}{36}+\frac{\sqrt{15}}{30}$	$\frac{2}{9}-\frac{\sqrt{15}}{15}$	$\frac{5}{36}$
	$\frac{5}{18}$	$\frac{4}{9}$	$\frac{5}{18}$

(c) Gauss (order 6)

0	$\frac{1}{9}$	$\frac{-1-\sqrt{6}}{18}$	$\frac{-1+\sqrt{6}}{18}$
$\frac{6-\sqrt{6}}{10}$	$\frac{1}{9}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{88-43\sqrt{6}}{360}$
$\frac{6+\sqrt{6}}{10}$	$\frac{1}{9}$	$\frac{88+43\sqrt{6}}{360}$	$\frac{88-7\sqrt{6}}{360}$
	$\frac{1}{9}$	$\frac{16+\sqrt{6}}{36}$	$\frac{16-\sqrt{6}}{36}$

(d) Radau-1A (order 5)

Table 1: The coefficients of fully implicit, three-stage Runge Kutta methods implemented in KPP–2.2.

$\frac{2-\sqrt{2}}{2}$	$\frac{2-\sqrt{2}}{2}$	0
1	$\frac{\sqrt{2}}{2}$	$\frac{2-\sqrt{2}}{2}$
	$\frac{\sqrt{2}}{2}$	$\frac{2-\sqrt{2}}{2}$

(a) Sdirk-2a (order 2, stiffly accurate)

$\frac{2+\sqrt{2}}{2}$	$\frac{2+\sqrt{2}}{2}$	0
1	$-\frac{\sqrt{2}}{2}$	$\frac{2+\sqrt{2}}{2}$
	$-\frac{\sqrt{2}}{2}$	$\frac{2+\sqrt{2}}{2}$

(b) Sdirk-2b (order 2, stiffly accurate)

$\frac{3-\sqrt{3}}{6}$	$\frac{3-\sqrt{3}}{6}$	0	0
$1-\frac{\sqrt{3}}{3}$	$\frac{3-\sqrt{3}}{6}$	$\frac{3-\sqrt{3}}{6}$	0
1	$\frac{3-\sqrt{3}}{6}$	$\frac{\sqrt{3}}{3}$	$\frac{3-\sqrt{3}}{6}$
	$\frac{3-\sqrt{3}}{6}$	$\frac{\sqrt{3}}{3}$	$\frac{3-\sqrt{3}}{6}$

(c) Sdirk-3a (order 2, stiffly accurate)

Table 2: The coefficients of second order SDIRK methods implemented in KPP–2.2.

```
#INTEGRATOR SDIRK
```

The selection of a particular method within this family is done via the input control vector entry ICNTRL(3) (the values 0 or 1 select Sdirk-2a, 2 selects Sdirk-2b, 3 selects Sdirk-3a, 4 selects Sdirk-4a, and 5 selects Sdirk-4b).

Using the implementations of the forward routines, integrators for the tangent linear model (TLM) and adjoint model (ADJ) were developed. Each family of forward methods (implicit Runge Kutta, singly-diagonally-implicit Runge Kutta, and Rosenbrock methods) was extended to calculate the TLM, the generated result is the sensitivity matrix.

3 Tangent Linear Models of Runge Kutta Methods

Small perturbations of the solution of (1) due to small changes δy^0 in the initial conditions propagate forward in time according to the sensitivity equation (also called the "*tangent linear model*")

$$\delta y' = J\,(t, y) \cdot \delta y \ , \quad t^0 \leq t \leq t^{\mathrm{F}} \ , \quad \delta y\left(t^0\right) = \delta y^0 \ , \quad \delta y(t) \in \Re^n \ . \tag{10}$$

Note that for nonlinear systems the sensitivity equations (10) depend on the solution of the chemical equations (1) through the Jacobian $J(t, y)$. Thus (1) and (10) have to be solved together to obtain a solution of the sensitivity equations.

A numerical solution of (10) can be obtained by applying a Runge Kutta method (2) to both (1) and (10). Alternatively, one can use variational calculus to compute the variations (sensitivities) of the solution of (2) due to small changes in the initial conditions. The two approaches are equivalent and they lead to the tangent linear Runge Kutta methods:

$$\begin{aligned} y^{n+1} &= y^n + h \sum_{i=1}^{s} b_i k_i \ , & \delta y^{n+1} &= \delta y^n + h \sum_{i=1}^{s} b_i \ell_i \ , \\ Y_i &= y^n + h \sum_{j=1}^{s} a_{ij} k_j \ , & \delta Y_i &= \delta y^n + h \sum_{j=1}^{s} a_{ij} \ell_j \ , \\ k_i &= f\,(\,T_i,\, Y_i\,) \ , & \ell_i &= J\,(T_i,\, Y_i) \cdot \delta Y_i \ . \end{aligned} \tag{11}$$

The numerical solution of the sensitivity part depends on the stage vectors of the nonlinear chemical system through $J\,(T_i,\, Y_i)$. Thus one has to solve simultaneously for the concentrations and their sensitivities.

For implementation purposes we use the same transformation as for the forward integrators and work with the sensitivity stage variables $\delta Z_i = \delta Y_i - \delta y^n$. The method (11) leads to a system of linear equations in the unknowns δZ_i

$$\delta Z_i - h \sum_{j=1}^{s} a_{ij}\, J\,(T_j,\, Y_j) \cdot \delta Z_j = h \sum_{j=1}^{s} a_{ij}\, J\,(T_j,\, Y_j) \cdot \delta y^n \ , \quad i = 1, \cdots, s \ . \tag{12}$$

For SDIRK methods the system (12) reduces to independent n-dimensional linear systems that can

be solved successively for each stage $i = 1, \cdots, s$

$$\Big[I_n - h\,\gamma\,J\,(T_i,\,Y_i)\Big]\cdot\delta Z_i = h\sum_{j=1}^{i-1} a_{ij}\,J\,(T_j,\,Y_j)\cdot\Big(\delta y^n + \delta Z_j\Big) + h\,\gamma\,J\,(T_i,\,Y_i)\cdot\delta y^n\ . \tag{13}$$

The KPP-2.2 implementation offers the option to solve the linear systems (13) directly, at the expense of one additional sparse LU decomposition per stage of matrices $I_n - h\gamma J\,(T_i, Y_i)$. The implementation also offers the alternative to solve (13) by quasi-Newton iterations of the form

$$\begin{aligned}\Big[I_n - h\,\gamma\,J\,(t^n,\,y^n)\Big]\cdot\Delta\delta Z_i^{[m]} &= \delta Z_i^{[m]} - h\sum_{j=1}^{i-1} a_{ij}\,J\,(T_j,\,Y_j)\cdot\Big(\delta y^n + \delta Z_j^{[m]}\Big)\\ &\quad -h\,\gamma\,J\,(T_i,\,Y_i)\cdot\Big(\delta y^n + \delta Z_i^{[m]}\Big)\\ \delta Z_i^{[m+1]} &= \delta Z_i^{[m]} - \Delta\delta Z_i^{[m]}\ ,\quad m = 0, 1, \cdots\end{aligned} \tag{14}$$

The equations (14) re-use the sparse LU factorization of the matrix $I_n - h\gamma J\,(t^n, y^n)$ which is available after advancing the concentrations by one step. This approach is closer in the spirit to the direct decoupled method [15] for sensitivity analysis.

The selection of the tangent linear SDIRK family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

```
#INTEGRATOR SDIRK_tlm
```

and the selection of a particular method within this family is done via the input control vector ICNTRL.

For fully implicit Runge Kutta methods KPP-2.2 offers the option to construct the $ns \times ns$ linear system (12) explicitly, and solve it using a direct full linear algebra method. As an alternative KPP-2.2 also offers the option of solving (12) by quasi-Newton iterations of the form:

$$\begin{aligned}\Delta\delta Z_i^{[m]} - h\sum_{j=1}^{s} a_{ij}\,J\,(t^n,\,y^n)\cdot\Delta\delta Z_j^{[m]} &= \delta Z_i^{[m]} - h\sum_{j=1}^{s} a_{ij}\,J\,(T_j,\,Y_j)\cdot\Big(\delta y^n + \delta Z_j^{[m]}\Big)\ ,\\ \delta Z_i^{[m+1]} &= \delta Z_i^{[m]} - \Delta\delta Z_i^{[m]}\ ,\quad m = 0, 1, \cdots\end{aligned} \tag{15}$$

Equation (15) defines a $ns \times ns$ linear system for the unknowns $\Delta\delta Z_1^{[m]}, \cdots, \Delta\delta Z_s^{[m]}$. It can be seen that the matrix of this linear system is $I_{ns} - hA \otimes J(t^n, y^n)$. The LU decomposition of this matrix is already available as it is also necessary in the calculation of concentrations using equation (9). The re-use of the main LU decomposition in the concentration and in the sensitivity solutions is similar in spirit to the direct-decoupled method [15].

The selection of the tangent linear fully-implicit Runge Kutta family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

```
#INTEGRATOR Runge_Kutta_tlm
```

and the selection of a particular method within this family is done via the input control vector ICNTRL.

We note that the direct-decoupled approach is relatively inexpensive since it re-uses the available LU decompositions. However the control of the error within the iterations (14) and (15) for sensitivities is challenging, due to the fact that sensitivities take values spanning many orders of magnitude. It is quite difficult in practice to set the absolute tolerances for sensitivities to appropriate values, and correctly define an error norm used to control the iterative process.

4 Discrete Adjoint Models of Runge Kutta Methods

4.1 Continuous Adjoint Models

The solution of inverse problems involving chemical kinetic systems (e.g., parameter fitting, optimal control, and data assimilation) require the minimization of a cost functional defined in terms of the chemical concentrations. Without loss of generality any inverse problem can be formulated as the following optimization problem: find the initial conditions for which a function of the system state at the final time is minimized,

$$\min_{y^0} \overline{\Psi}\left(y^0\right) = h\Big(y(t^{\mathrm{F}})\Big) \quad \text{subject to (1)} \ . \tag{16}$$

To apply a gradient based optimization procedure one needs to compute the derivatives of the cost function $\overline{\Psi}$ with respect to the initial conditions. It is well known [26] that these derivatives can be obtained efficiently by solving the continuous adjoint equation

$$\lambda' = -J^T\Big(t, y(t)\Big)\,\lambda \ , \quad \lambda\left(t^{\mathrm{F}}\right) = \frac{\partial h}{\partial y}\Big(y(t^{\mathrm{F}})\Big) \ , \quad t^{\mathrm{F}} \geq t \geq t^0 \tag{17}$$

backwards in time from t^{F} to t^0 to obtain

$$\lambda(t^0) = \frac{\partial \overline{\Psi}}{\partial y^0}.$$

Note that the continuous adjoint equation (17) is formulated based on the forward solution $y(t)$.

In the *continuous adjoint* approach one solves the equation (17) backward in time with a Runge Kutta method (2) with coefficients $\tilde{a}_{i,j}$, $\tilde{b}_i$, $\tilde{c}_i$ to obtain

$$\begin{aligned} \lambda^n &= \lambda^{n+1} + h \sum_{i=1}^{s} \tilde{b}_i \, \tilde{\ell}_i \ , \\ \tilde{\ell}_i &= J^T\Big(t^{n+1} - \tilde{c}_i h,\, y(t^{n+1} - \tilde{c}_i h) \Big) \, \lambda^i \ , \quad \lambda^i = \lambda^{n+1} + h \sum_{j=1}^{s} \tilde{a}_{i,j} \tilde{\ell}_j \ . \end{aligned} \tag{18}$$

The terminal value of the adjoint variable $\lambda(t^0)$ is an approximation of the gradient of (16).

The continuous approach (18) requires the values of the forward model solution at intermediate times $y(t^{n+1} - \tilde{c}_i h)$. These values are obtained by interpolation from the forward solution, which is saved in checkpoints at each forward time step t^n. It is our experience that in practice the interpolation procedure works properly only when the forward solution is stored often; in other words, only when the step size in the forward solution is small, typically much smaller than that required by the accuracy constraints. For this reason we have not implemented the continuous adjoint approach in KPP–2.2.

4.2 Discrete Adjoint Models

In practice the equation (1) is solved numerically on a computer. A Runge Kutta discretization (2) advances the solution in time as follows

$$y^{n+1} = \mathcal{M}_n \left(y^n \right) \ , \quad y^N = \mathcal{M}_{N-1}\Big(\mathcal{M}_{N-2}\big(\cdots \mathcal{M}_0(y^0) \big) \Big) \ , \tag{19}$$

where $\mathcal{M}$ symbolically denotes one step of the method (2), $t^N = t^{\mathrm{F}}$ and the numerical solution is $y^n \approx y(t^{\mathrm{n}})$. The optimization problem (16) is formulated in terms of the numerical solution minimized,

$$\min_{y^0} \Psi\left(y^0 \right) = h\Big(y^N \Big) \quad \text{subject to (19)} \ . \tag{20}$$

To estimate the gradient of the cost function (20) several approaches are possible.

In the *discrete adjoint* approach the gradient of (16) is computed directly from (19) using the transposed chain rule

$$\left(\frac{d\Psi}{dy^0} \right)^T = \left(\frac{d\mathcal{M}_0}{dy^0}(y^0) \right)^T \cdots \left(\frac{d\mathcal{M}_{N-1}}{dy^{N-1}}(y^{N-1}) \right)^T \left(\frac{dh}{dy^N}(y^N) \right)^T \ .$$

This calculation proceeds backwards in time, i.e. the expression is evaluated right to left as follows

$$\lambda^N = \left(\frac{dh}{dy^N}(y^N) \right)^T \dots \quad \lambda^n = \left(\frac{d\mathcal{M}_n}{dy^n}(y^n) \right)^T \lambda^{n+1} \dots \quad \lambda^0 = \left(\frac{d\Psi}{dy^0} \right)^T \ . \tag{21}$$

We will call λ^n discrete adjoint variables. Their evaluation requires the forward numerical solution y^0 to y^N to be available during the backward calculation.

Discrete adjoints are useful in optimization since they provide the gradients of the numerical function that is being minimized. Moreover, they can be calculated by reverse mode automatic differentiation.

In KPP–2.2 we have implemented the discrete adjoints of Runge Kutta methods. In [30] we have shown that the discrete Runge Kutta adjoint (21) can be regarded as a new numerical method applied to the continuous adjoint equation (17). The main results of [30] are:

- the discrete adjoint of a Runge Kutta method of order p is an order p discretization of the continuous adjoint equation (17);
- consider a singular perturbation problem and a cost functional defined only in terms of the nonstiff variable. The discrete adjoint of an L-stable Runge Kutta method with an invertible coefficient matrix A produces solutions of the same accuracy as the continuous adjoint approach.

These two theoretical properties imply that discrete Runge Kutta adjoints are very well suited for solving inverse problems with stiff chemical systems. This conclusion is also supported by other previous studies. Consistency properties of discrete Runge Kutta adjoints have been studied by Hager [19] in the context of control problems. Walther [36] has studied the effects of reverse mode automatic differentiation on explicit Runge Kutta methods. Giles [17] has discussed Runge Kutta adjoints in the context of steady state flows.

Hager [19] has shown that one step of the discrete adjoint of the Runge Kutta method (2) reads

$$\begin{aligned} u_i &= h\, J^T(T_i, Y_i) \cdot \left(b_i \lambda^{n+1} + \sum_{j=1}^{s} a_{j,i} u_j \right) \ , \quad i = s, \cdots, 1 \\ \lambda^n &= \lambda^{n+1} + \sum_{j=1}^{s} u_j \ . \end{aligned} \tag{22}$$

For $b_i \neq 0$ the RK adjoint (22) can be rewritten as another Runge Kutta method [19] applied to the

continuous adjoint equation (17)

$$\begin{aligned} \lambda^n &= \lambda^{n+1} + h \sum_{i=1}^{s} \overline{b}_i \, \ell_i \ , \quad \ell_i = J^T \Big(t_{n+1} - \overline{c}_i h, \, Y_{s+1-i} \Big) \, \Lambda_i \ , \\ \Lambda_i &= \lambda^{n+1} + h \sum_{j=1}^{s} \overline{a}_{i,j} \ell_j \ , \end{aligned} \tag{23}$$

$$\text{where} \qquad \overline{b}_i = b_{s+1-i} \ , \quad \overline{c}_i = 1 - c_{s+1-i} \ , \quad \overline{a}_{i,j} = \tfrac{a_{s+1-j,s+1-i} \cdot b_{s+1-j}}{b_{s+1-i}}$$

We will call (23) the *formal discrete adjoint method* of (2). Note the similarity between (23) and (18). The only difference is that the Jacobian in (23) is evaluated at the stage vector Y_{s+1-i}, while in (18) is evaluated at the interpolated solution vector $y(t^{n+1} - \tilde{c}_i h)$. To some extent the properties of the discrete adjoint are in this case the properties of the formal adjoint method (23).

The formal adjoints $(\overline{A}, \overline{b}, \overline{c})$ of the Runge Kutta methods (A, b, c) implemented in KPP–2.2 can be easily derived using the transformation (23) and are as follows:

- the formal adjoint of Radau-2A is Radau-1A; the formal adjoint of Radau-1A is Radau-2A;
- the formal adjoint of Lobatto-3C is Lobatto-3C (i.e., Lobatto-3C is formally self-adjoint). Therefore its formal adjoint is stiffly accurate;
- Gauss is formally self-adjoint;
- Sdirk-3a is formally self-adjoint (and therefore its formal adjoint is stiffly accurate);
- The formal adjoints of Sdirk-2a, Sdirk-2b, Sdirk-4a, and Sdirk-4b are other SDIRK methods.

4.3 Implementation Aspects

Our implementation of the discrete adjoints in KPP–2.2 follows the formulation (22). This equation defines a linear system of dimension $ns \times ns$ in the stage vectors $u_1, \cdots, u_s$

$$u_i - h \, J^T \left(T_i, Y_i \right) \cdot \sum_{j=1}^{s} a_{j,i} u_j = h \, b_i \, J^T \left(T_i, Y_i \right) \cdot \lambda^{n+1} \ , \quad i = 1, \cdots, s \ . \tag{24}$$

For SDIRK methods the system (24) decouples into s systems of dimension $n \times n$, which can be solved successively for each stage (from the last to the first)

$$\Big[I_n - h \, J \left(T_i, Y_i \right) \Big]^T \cdot u_i = h \, J^T \left(T_i, Y_i \right) \cdot \Big(b_i \, \lambda^{n+1} + \sum_{j=i+1}^{s} a_{j,i} u_j \Big) \ , \quad i = s, s-1, \cdots, 1 \ . \tag{25}$$

Our KPP–2.2 implementation offers two options. The linear systems (25) can be solved directly, at the expense of one LU factorization per stage of the matrices $I_n - h\,J\,(T_i, Y_i)$, or via iterations

$$\begin{aligned} \Big[I_n - h\,J\,(t^n, y^n)\Big]^T \cdot \Delta u_i^{[m]} &= h\,J^T\,(T_i, Y_i) \cdot \Big(b_i\,\lambda^{n+1} + \sum_{j=1}^{s} a_{j,i} u_j^{[m]}\Big)\,, \\ u_i^{[m+1]} &= u_i^{[m]} - \Delta u_i^{[m]} \end{aligned} \tag{26}$$

which re-use the LU factorization of $I_n - h\,J\,(t^n, y^n)$ for all the stages. The iterations (26) are similar to the ones for tangent linear calculations (14).

The selection of the discrete adjoint SDIRK family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

```
#INTEGRATOR SDIRK_adj
```

and the selection of a particular method within this family is done via the input control vector ICNTRL.

For fully implicit Runge Kutta methods the KPP–2.2 implementation offers the option to construct the system (24) explicitly and solve it by a direct $ns \times ns$ LU factorization. The implementation also offers the option of solving (24) by iterations of the form

$$\begin{aligned} \Big[I_{ns} - hA \otimes J(t^n, y^n)\Big]^T\ \Delta U^{[m]} &= \begin{bmatrix} u_1^{[m]} - h\,J^T(T_1, Y_1) \cdot \Big(b_1\,\lambda^{n+1} \sum_{j=1}^{s} a_{j,1} u_j^{[m]}\Big) \\ \vdots \\ u_s^{[m]} - h\,J^T(T_s, Y_s) \cdot \Big(b_s\,\lambda^{n+1} + \sum_{j=1}^{s} a_{j,s} u_j^{[m]}\Big) \end{bmatrix} \\ U^{[m+1]} &= U^{[m]} - \Delta U^{[m]}\ . \end{aligned} \tag{27}$$

The matrix $I_{ns} - hA \otimes J(t^n, y^n)$ in (27) is the same matrix used in the solution of the forward model (9), as well as in the iterative solution of the tangent linear model (15). Its LU factorization can be saved from the forward simulation and reused in adjoint.

The selection of the adjoint fully-implicit Runge Kutta family of integrators in KPP–2.2 is done by the using following command in the model (*.kpp) file:

```
#INTEGRATOR Runge_Kutta_adj
```

and the selection of a particular method within this family is done via the input control vector ICNTRL.

We note that the iterative approaches (26) and (27) to solve the linear adjoint systems are relatively inexpensive since they re-use the available LU decompositions. However the control of the error within the

iterations (26) and (27) for adjoints is challenging, due to the fact that adjoints (like forward sensitivities) take values spanning many orders of magnitude. It is quite difficult in practice to set the adjoint absolute tolerances to appropriate values.

5 The DECLARE option

We now report on a new option that has been implemented in KPP–2.2. The command

```
#DECLARE [ symbol | value ]
```

in the model (*.kpp) file tunes the form of the array declarations in the generated code.

The default `#DECLARE symbol` option dimensions the arrays in the generated code in terms of the model parameters like the number of variables and the number of reactions, e.g.,

```
USE  model_Parameters
REAL(kind=dp) :: VAR(NVAR), RCONST(NREACT)
```

The values of the constants like `NVAR`, `NREACT` are declared in the Parameters module (or header file).

The option `#DECLARE value` dimensions the arrays in the generated code by using the model parameter values directly, e.g.,

```
REAL(kind=dp) :: VAR(74), RCONST(211)
```

This option alleviates the cross-dependencies of the generated code (here, the dependency on the Parameters module, or header file, has been removed).

6 Numerical Results

We now illustrate the application of the new numerical integrators on the chemical model SAPRC99 [11], which is widely used in real atmospheric chemistry applications. The SAPRC99 mechanism consists of 74 active chemical species interacting in 211 chemical reactions. We consider emissions of key species being added to the chemical rates of transformation. The initial conditions are obtained after a 24 hour integration of the system with emissions; this initial period allows the system to evolve past the

initial transients and develop quasi-steady-states. The integration interval is 24 hours (beyond the initial time); the accuracy of the numerical solutions is assesses at the end of the simulation interval. Reference solutions of concentrations and sensitivities are obtained with the SEULEX stiff differential equation solver of Hairer and Wanner [21] with very tight accuracies ($RTOL = 10^{-12}$, $ATOL = 10^{-8}$). SEULEX uses an extrapolation formula and offers an independent approach to benchmark the accuracy of KPP–2.2 Runge Kutta integrators.

6.1 Forward Methods and Integration Options

Three families of stiff integration methods are implemented in KPP–2.2: fully implicit Runge Kutta, SDIRK, and Rosenbrock. Each family has a different implementation. The selection switch ICNTRL(3) allows the user to choose a particular method within each family. The major selection switch in the Runge Kutta family chooses between Radau-2A, Lobatto-3C, Gauss, and Radau-1A. For SDIRK integration, the same switch is used to select between Sdirk-2a, Sdirk-2b, Sdirk-3a, Sdirk-4a, or Sdirk-4b. The available choices of Rosenbrock methods are Ros-2, Ros-3, Ros-4, Rodas-3, and Rodas-4.

Some control options are general and apply to all families. The relative and absolute error tolerances can be a scalar or a vector (different tolerance for each species). The maximum number of integration steps before unsuccessful return can be specified to limit the return time for non-converging calculations. Various other switches select specialized options for the different integration families.

6.1.1 Fully Implicit Runge Kutta Options

We now discuss the most important switches that allow the user to tune the behavior of the fully implicit Runge Kutta family.

Tuning the Newton iterations. Various internal coefficients and parameters for Newton iteration can be specified. They include the maximum number of Newton iterations, stopping criterion (based on estimating the magnitude of iteration error), bounds on step decrease, increase, and on step rejection. Default values are assigned to these parameters after careful experimenting and using [21]. The choice of starting values for Newton iteration are zero or the extrapolated collocation solution. As seen in Figure

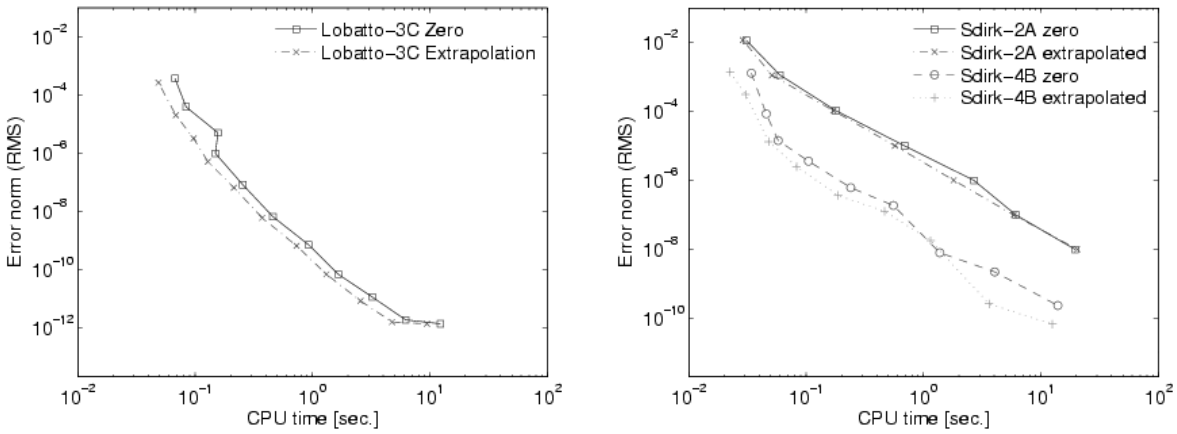

Figure 2: Work-precision diagram (accuracy vs. CPU-time) for the forward methods. Shown are Lobatto-3C (left) and Sdirk-2a and Sdirk-4b (right) results when the starting values for Newton iterations are taken equal to zero and when they are obtained by extrapolation.

2 (left) for Lobatto-3C, the CPU time decreases when using extrapolation. This effect is clearly due to the fewer number of Newton iterations necessary to reach the desired accuracy.

Step size control. Two step-size strategies are implemented: the classical approach and the modified predictive controller (Gustafsson, [18]). The Gustafsson predictive controller takes a few more steps but secures the computation by using a more precautions approach. If the predicted change in step size is small then the code keeps the same time step and attempts to re-use the LU factorizations of the previous step.

Error estimation. Two different error estimation strategies are implemented. The first one is the classical error estimation [21] which uses an embedded third order method constructed based on an additional explicit stage (at the beginning of the time step). The embedded solution is:

$$\hat{y}^{n+1} = y^n + h\left(\hat{b}_0 f(t^n, y^n) + \sum_{i=1}^{3} \hat{b}_i f(t^n + c_i h, y^n + Z_i)\right) \tag{28}$$

Our newly implemented error estimation uses two additional stages: an explicit stage (at the beginning of the time step) and another SDIRK stage which re-uses the real LU decomposition from the solution

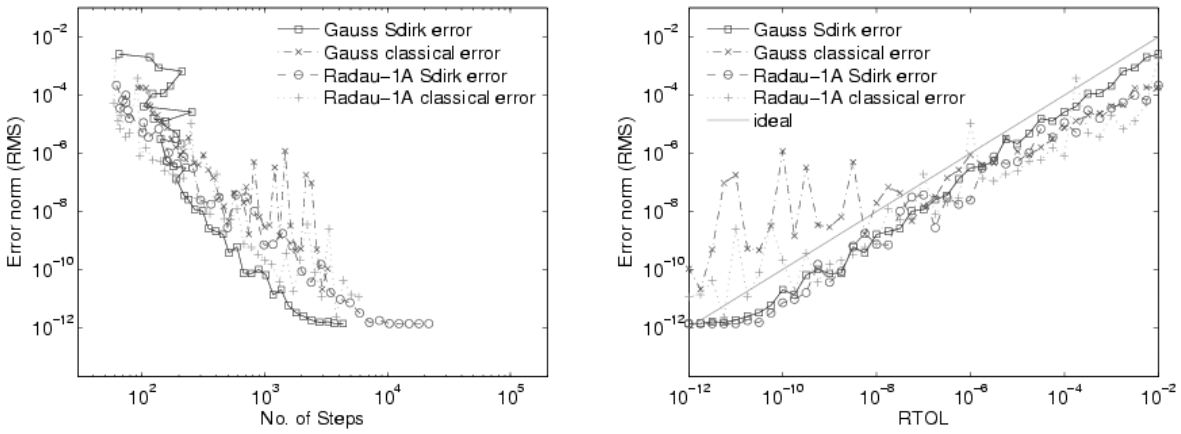

Figure 3: Comparison of forward Gauss and Radau-1A methods using classical versus SDIRK error estimation. The comparison includes the total number of steps used (left) and the solution accuracy (right).

of the main method. The embedded solution reads:

$$\hat{y}^{n+1} = y^n + h\left(\hat{b}_0 f(t^n, y^n) + \sum_{i=1}^{3} \hat{b}_i f(t^n + c_i h, y^n + Z_i) + \gamma f(t^n + h, y^n + Z_4)\right) \tag{29}$$

All the embedded methods have been chosen to be stiffly accurate and have order 3. The coefficients of the additional stage are given in Appendix A. The results shown in Figure 3 indicate that the new error estimator works well, and brings marked improvements for Gauss and Radau-1A. While the classical error estimator does not predict the error very well for low relative tolerances, the Sdirk error estimation provides very good results and keeps the accuracy of the solution below the ideal line.

6.1.2 SDIRK Options

The SDIRK solution uses Newton iterations; the SDIRK implementation offers similar options as the implicit Runge Kutta methods described above. Figure 2 (right) shows the difference of using zero or extrapolation starting values for Newton iteration on methods Sdirk-2a and Sdirk-4b. While Sdirk-2a only shows slight changes, the larger differences for Sdirk-4b are due to different step size selections (when starting values are set to zero they may reach the maximum number of Newton iterations due to the large resulting error). The classical error estimation [21] based on an embedded solution with different weights has been implemented and works well.

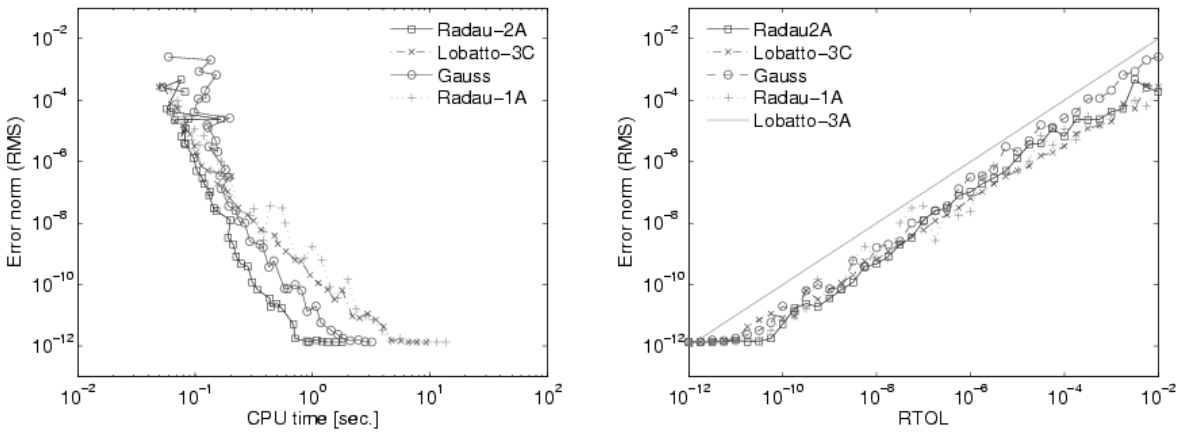

Figure 4: Accuracy vs. CPU-time (left) and accuracy vs. relative tolerance (right) for forward fully implicit Runge Kutta methods.

6.1.3 Comparison of Forward Integrator Families

All new solvers have been tested thoroughly. Figures 4–6 (left) show the work precision diagrams (solution accuracy versus CPU time) for all the forward methods implemented in KPP–2.2. The implicit Runge Kutta integrators were tested using the relative tolerance range RTOL $\in [10^{-2}, 10^{-12}]$ and SDIRK and Rosenbrock integrators are presented on tolerance range RTOL $\in [10^{-2}, 10^{-10}]$. The absolute error tolerances are set to ATOL $= 10^5 \cdot$ RTOL. Figures 4–6 present the accuracy of the computed solution versus the requested RTOL; this illustrates the quality of the error control (an ideal error controller would yield the accuracy very near the user specified tolerance). For fully implicit Runge Kutta methods the new error estimator using an additional SDIRK stage was used. Gustafsson predictive error controller was selected for all integrators. The starting values of Newton iterations were interpolated (the default setting) for both Runge Kutta and SDIRK families.

Figure 7 compares the work-precision diagrams of the best methods of each family (Radau-2A, Sdirk-4b, and Rodas-4). While the Rosenbrock method is the best for low tolerances, the fully implicit Runge Kutta is to be preferred for high accuracy calculations.

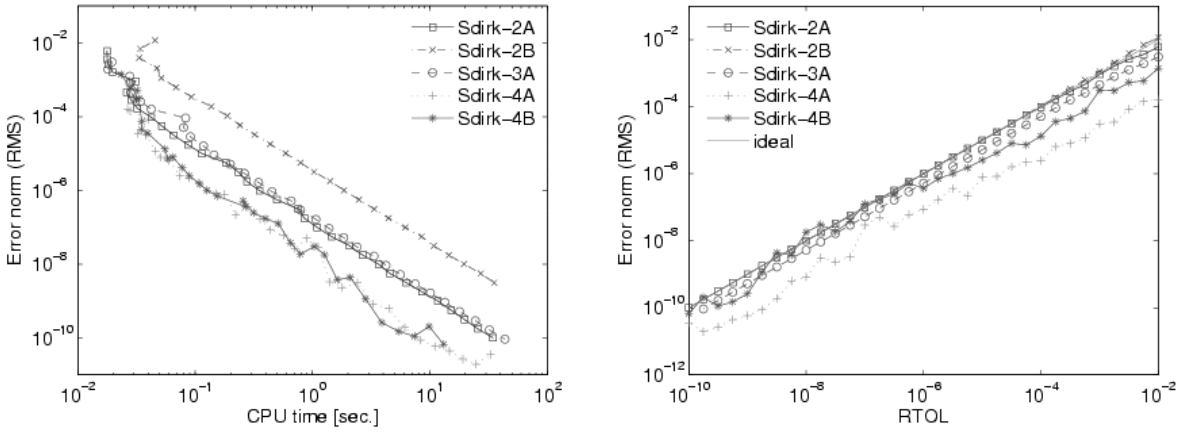

Figure 5: Accuracy vs. CPU-time (left) and accuracy vs. relative tolerance (right) for forward Sdirk methods.

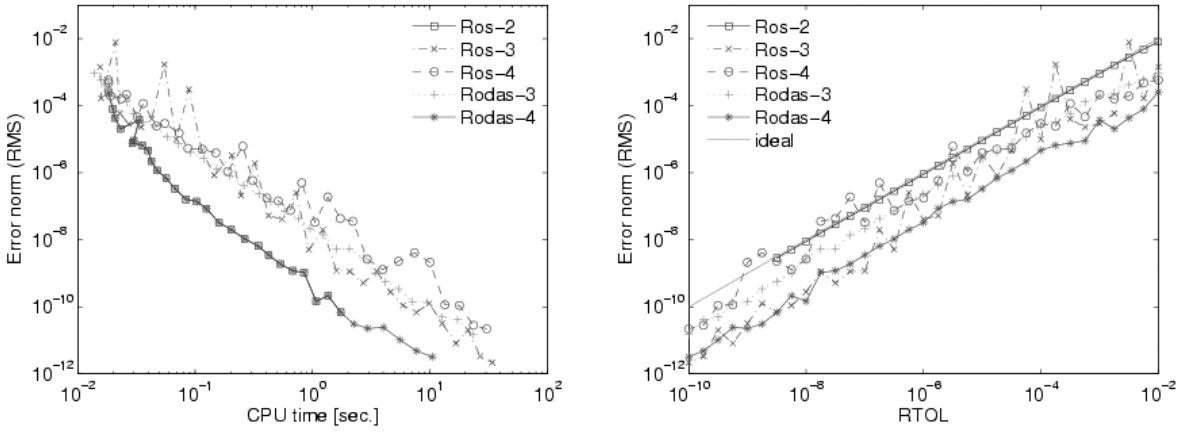

Figure 6: Accuracy vs. CPU-time (left) and accuracy vs. relative tolerance (right) for forward Rosenbrock methods.

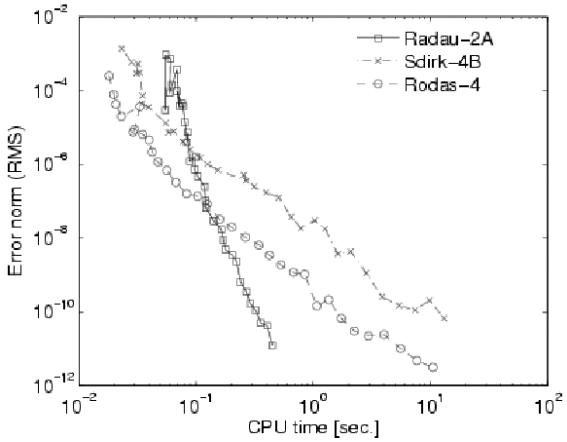

Figure 7: Accuracy versus CPU-time for Radau-2A, Sdirk-4b, and Rodas-4 forward methods.

6.2 Tangent Linear Methods and Integration Options

The tangent linear model (TLM) implementations advance in time both the concentrations and the forward sensitivity coefficients. The TLM of each method for all the forward integration families is implemented in KPP–2.2. The options of the forward code are also available in TLM routines. In addition several TLM-specific options are available to the user as follows:

- apply forward error estimation only (there is no explicit control over the accuracy of the sensitivities);
- control the convergence of TLM Newton iterations; and
- control the truncation errors for both the concentrations and the TLM sensitivities (and use both in step-size selection).

In these experiments we look at the sensitivity coefficients of 10 long-lived species with respect to their initial values. To be specific, we compute $\partial y_i(t^f)/\partial y_j(t^0)$ for all i, j in a subset of 10 long lived species. The relative errors of TLM variables are computed for these 100 sensitivity coefficients; the reference solution is obtained by running SEULEX on the TLM differential equations (10).

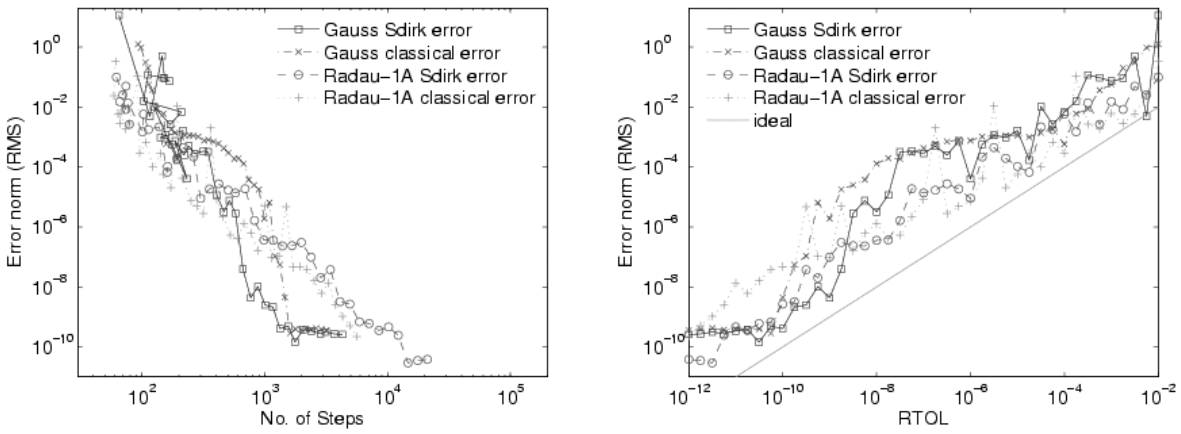

Figure 8: Comparison of tangent linear Gauss and Radau-1A methods using classical and SDIRK error estimates. Compared are the number of steps (left) and accuracy of the solution (right).

6.2.1 Tangent Linear Implicit Runge Kutta Methods Options

We have seen that in the forward integration the choice of the error estimation method influences the accuracy of the result. The forward integration method also influences the accuracy of the TLM integration as can be seen in Figure 8.

Controlling the convergence of Newton TLM iterations. There are two options implemented for the Newton TLM iterations. The first is to predefine the number of iterations for sensitivities as the number of Newton iterations for concentrations plus one. The second option is to use the relative and absolute tolerances for sensitivities to control the convergence of the Newton TLM iterations. Because sensitivity values are typically different than the concentration values one needs different absolute tolerances for the concentrations and for sensitivities. A comparison of these two options is shown in Figures 9 and 10. The control of Newton TLM iteration error leads to smoother curves, but considerably increases the number of steps. The number of Newton iterations in this case is smaller than the number of Newton iterations required by the forward method. We recommend the user to combine the control of Newton TLM iteration error with the control of the TLM truncation error (discussed below) in a fully adaptive code.

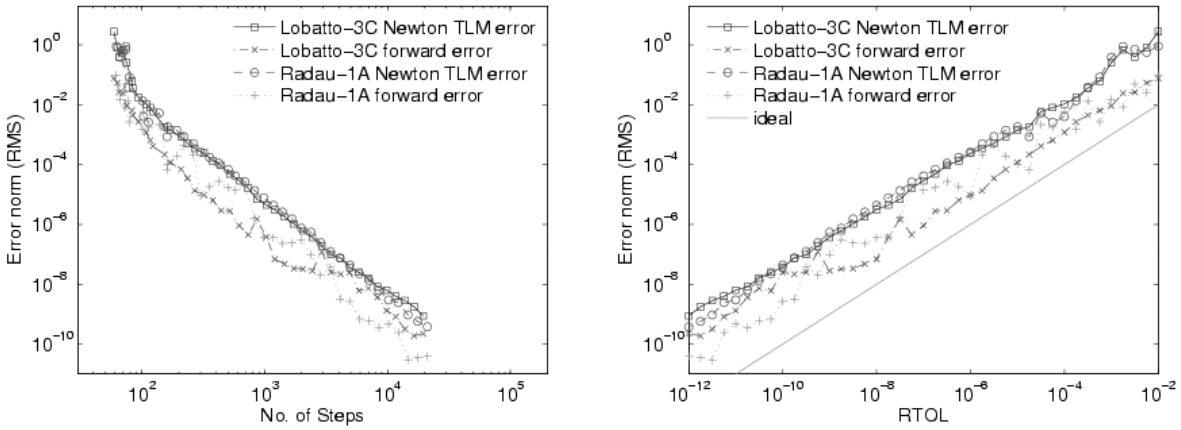

Figure 9: Comparison of steps (left) and accuracy (right) of tangent linear Lobatto-3C and Radau-1A using TLM Newton iteration convergence and forward iteration count.

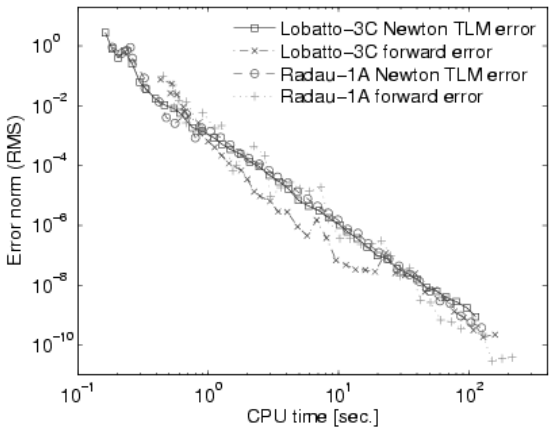

Figure 10: Comparison of CPU-time (left) and (right) of tangent linear Lobatto-3C and Radau-1A using TLM Newton iteration convergence and forward iteration count

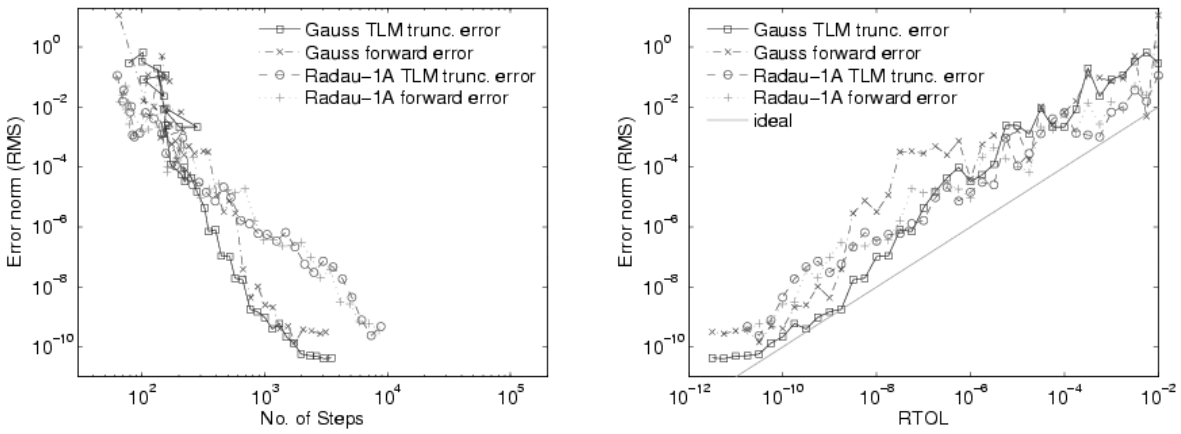

Figure 11: Accuracy vs. the number of steps steps (left) and accuracy vs. tolerance (right) for of TLM Gauss and Radau-1A methods. Results are shown for both step size control strategies (with and without TLM truncation error control).

Controlling the TLM truncation error. During the forward integration (the calculation of concentrations) estimates of the truncation error are used to decide whether the step is accepted or rejected, and to compute the next step size. Two options are implemented in KPP–2.2 for the calculation of sensitivities.

The first option is to use only the forward (concentration) error estimates for step size control. No error control is used for the TLM variables; the resulting sensitivities can be viewed as sensitivities of the Runge Kutta numerical solution (2) (rather than as approximations of solutions of the continuous tangent linear ODE 10).

The second option implemented in KPP–2.2 is to estimate the truncation errors for both the forward solution (concentrations) and the TLM solution (sensitivities). 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 used to control the step size. Results presented in Figures 11 and 12 indicate that while Radau-1A TLM behaves similarly, the truncation error forces higher accuracy for Gauss TLM.

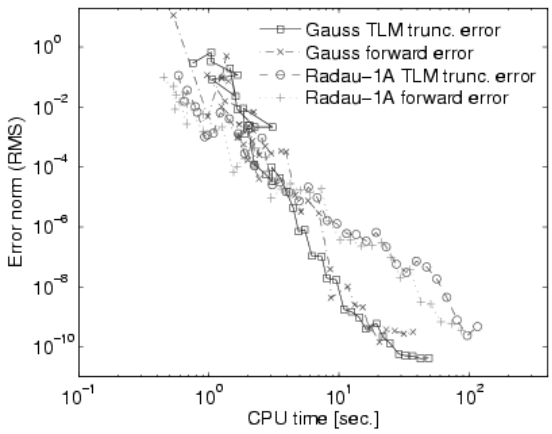

Figure 12: Accuracy vs. CPU-time for TLM Gauss and Radau-1A. Results are shown for both step size control strategies (with and without TLM truncation error control).

6.2.2 Tangent Linear Sdirk Method Options

For the Sdirk integrators an additional option has been implemented. The TLM solution can either be computed using modified Newton iterations that re-use the same LU decomposition or by computing the direct solution at the expense of an additional LU factorization per stage. The results presented in Figure 13 indicate that the same accuracy is reached faster with the direct methods.

As with fully implicit Runge Kutta TLM methods, for Sdirk TLM methods the user has the option to control the TLM Newton iteration convergence and/or estimate the TLM truncation error and use it in the step size decisions. The results of different options with TLM Sdirk-4A are shown in Figure 14.

6.2.3 Tangent Linear Rosenbrock Method Options

The options of using the TLM truncation error in step size control has been implemented with Rosenbrock TLM methods as well. Our experience is that using the forward error estimation for step size control is a good strategy; adding the TLM truncation error control not change the step size significantly.

6.2.4 Comparison of the Tangent Linear Integrator Families

Efficiency and accuracy results for the TLM integrators are shown in Figure 15 (TLM fully implicit Runge Kutta), Figure 16 (TLM SDIRK), and Figure 17 (TLM Rosenbrock methods). The fully Runge Kutta

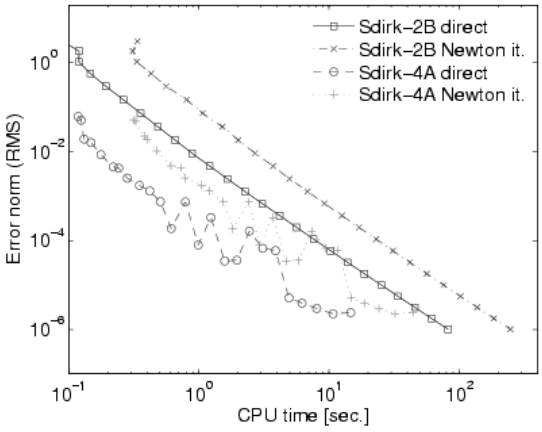

Figure 13: Accuracy vs. CPU-time for TLM Sdirk-2B and Sdirk-4A methods. Results are shown for both the direct and the iterative approaches to solve for TLM variables.

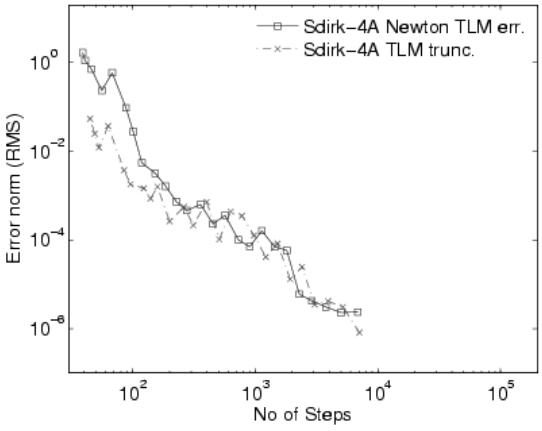

Figure 14: Comparison of tangent linear Sdirk-4A using TLM Newton iteration convergence and TLM truncation error estimation

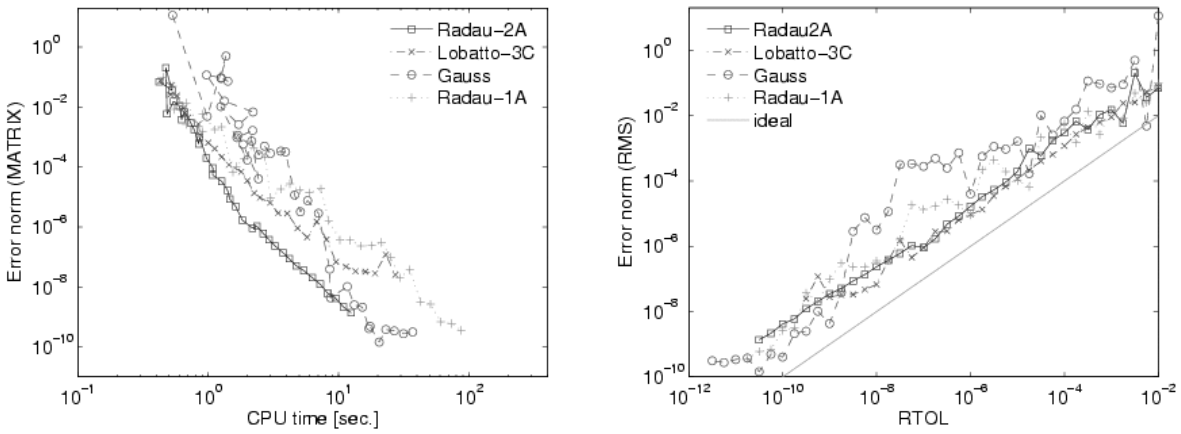

Figure 15: Accuracy vs. CPU-time (left) and accuracy vs. tolerance (right) for TLM fully implicit Runge Kutta methods.

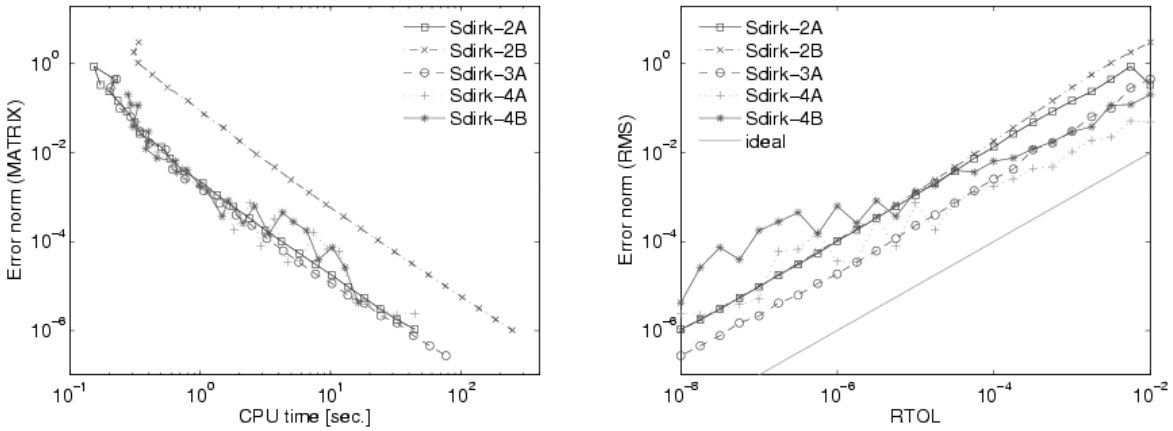

Figure 16: Accuracy vs. CPU-time (left) and accuracy vs. tolerance (right) for TLM SDIRK methods.

and SDIRK integrators were tested using relative tolerances in the range RTOL $\in [10^{-2}, 10^{-12}]$. For Sdirk and Rosenbrock integrators this range was limited to RTOL $\in [10^{-2}, 10^{-8}]$ due to the computationally intensive calculations for some of the methods. The absolute error tolerance was set to ATOL $= 10^5 \cdot$ RTOL. The TLM error estimation (for TLM Newton iterations and TLM truncation error) have been disabled. For Sdirk routines, modified Newton iterations have been used.

In Figure 18 we compare efficiency of several TLM methods, one from each family (Radau-2A, Sdirk-4B, and Rodas-4) within the relative tolerance range RTOL $\in [10^{-2}, 10^{-8}]$. Rodas-4 is the most efficient for low accuracies, while Radau-2A is the most efficient in the high accuracy range.

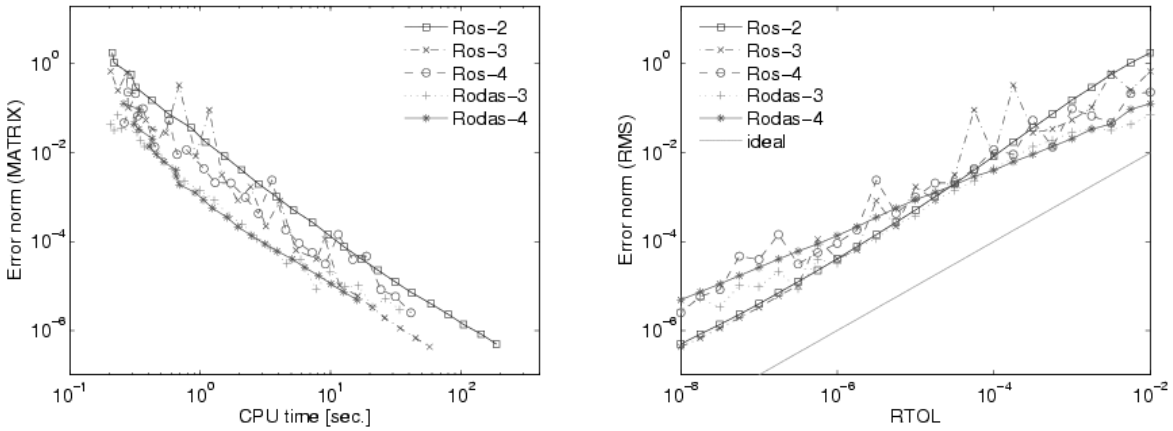

Figure 17: Accuracy vs. CPU-time (left) and accuracy vs. tolerance (right) for TLM Rosenbrock methods.

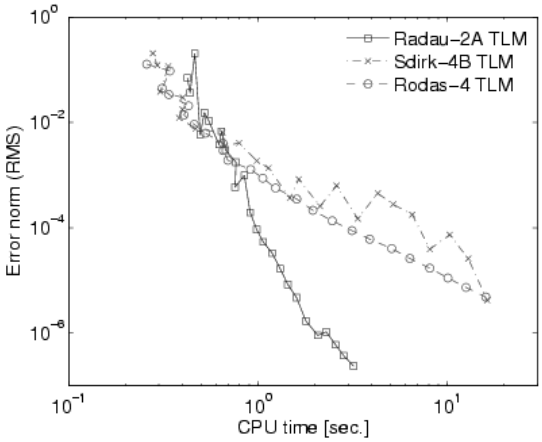

Figure 18: Accuracy vs. CPU-time for TLM Radau-2A, TLM Sdirk-4B, and TLM Rodas-4 methods.

6.3 Adjoint Methods and Integration Options

The adjoint experiments are carried out as follows. The forward code is run once from the initial time to the final time. At each step the time t^n, the step size h, the vector of concentrations y^n, and the intermediate stage vectors Z_i are all saved in checkpoints. The user has the option to store the LU decomposition of the relevant matrix ($I_{ns} - hA \otimes J$ for fully implicit or $I_n - h\gamma J$ for SDIRK methods).

The discrete adjoint method is then run backwards in time, and traces backwards the same sequence of steps as the forward method. At each step the concentrations, stage vectors, and the LU decomposition (if needed) are retrieved from the checkpoint storage. No source terms are added to the adjoint calculation (this means that the cost functional (20) is defined at the final time, without loss of generality). The calculation of the adjoint step then proceeds as discussed previously. The linear system for the adjoint stage variables is solved by Newton iterations. The user has the option to store the LU factorizations during the forward integration, and re-use them during the adjoint calculation for high computational efficiency. If they do not converge we do not have the option of reducing the stepsize and reiterating. If the Newton iterations do not converge the code switches automatically to a direct solution method (at the expense of additional LU decompositions).

Since the choice of the step sizes is done exclusively by the forward method, the accuracy of the adjoint solution will depend on the error control used during the forward integration. Figure 19 shows the impact on adjoint accuracy of of using the classical and the 2-stage SDIRK error estimation in the forward implicit Runge Kutta integration.

6.3.1 Comparison of the Adjoint Integrator Families

In the adjoint experiments we look at the same 100 sensitivity coefficients $\partial y_i(t^f)/\partial y_j(t^0)$ for all i, j in the subset of 10 long lived species. The reference values are the ones computed with SEULEX by the direct decoupled method.

The efficiency and accuracy graphs for adjoint integrators are shown in Figure 20 (adjoint fully implicit Runge Kutta), Figure 21 (adjoint SDIRK), and Figure 22 (adjoint Rosenbrock). All integrators were tested using the relative tolerance range RTOL $\in [10^{-2}, 10^{-8}]$ and absolute error tolerances set to

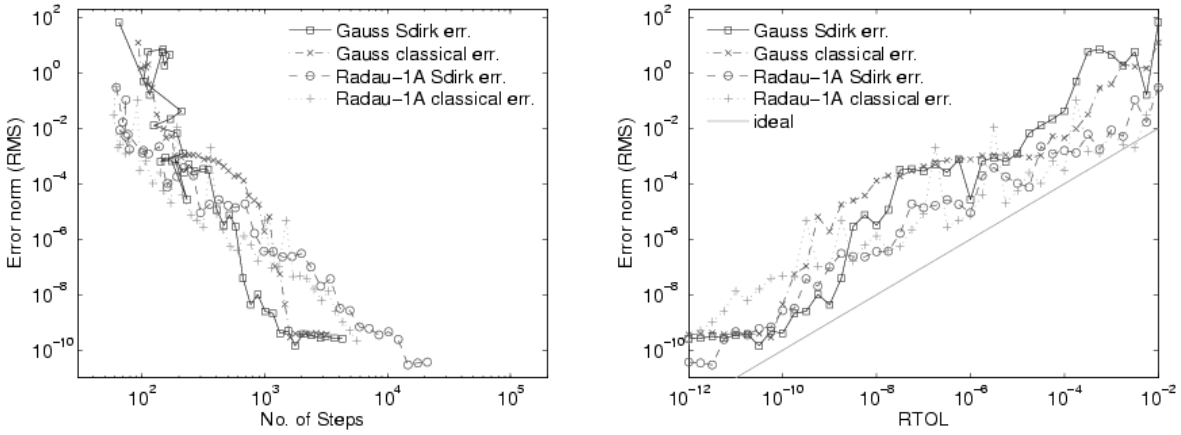

Figure 19: Comparison of adjoint Gauss and Radau-1A methods using classical and SDIRK error estimation in the forward calculation. Shown are the accuracy vs. the number of steps (left) and the accuracy vs. tolerance (right).

ATOL $= 10^3 \cdot$ RTOL. The same settings as in forward comparison are used to compute the forward solution (for SDIRK routines, modified Newton iterations have been used). The slopes of the work-precision diagrams show that the order of accuracy of discrete adjoint Runge Kutta methods is the same as the order of the forward Runge Kutta methods. The performance of methods representing each integrator families (adjoint Radau-2A, adjoint Sdirk-4B, and adjoint Rodas-4) are compared in Figure 23 using the relative tolerances RTOL $\in [10^{-2}, 10^{-8}]$. The same conclusion holds as for the direct and for the TLM comparisons: Rodas-4 is the most accurate method for adjoint calculations with low accuracy, while Radau-2A is the most accurate for high accuracy adjoint calculations.

7 Variational Data Assimilation

We now illustrate the use of discrete adjoint Runge Kutta methods in the solution of inverse problems. Specifically, we apply discrete Runge Kutta adjoint solutions to variational chemical data assimilation [13]. Four dimensional variational data assimilation (4D-VAR) approach adjusts the model initial conditions and model parameters to minimize the mismatch between the model predictions and the observations. The cost function gradients needed to solve the minimization problem are obtained by adjoint modeling.

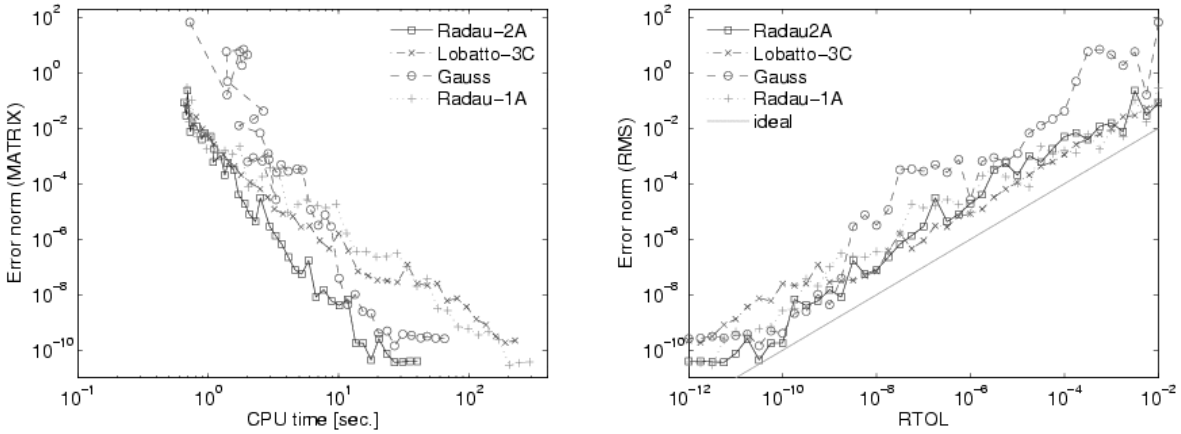

Figure 20: Accuracy vs. CPU time (left) and accuracy vs. tolerance (right) for adjoint fully implicit Runge Kutta methods.

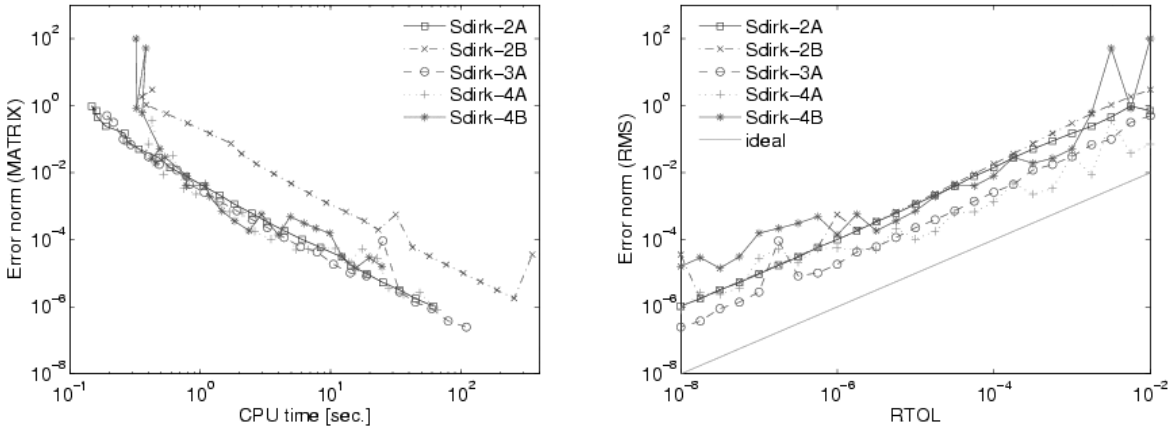

Figure 21: Accuracy vs. CPU time (left) and accuracy vs. tolerance (right) of adjoint SDIRK methods.

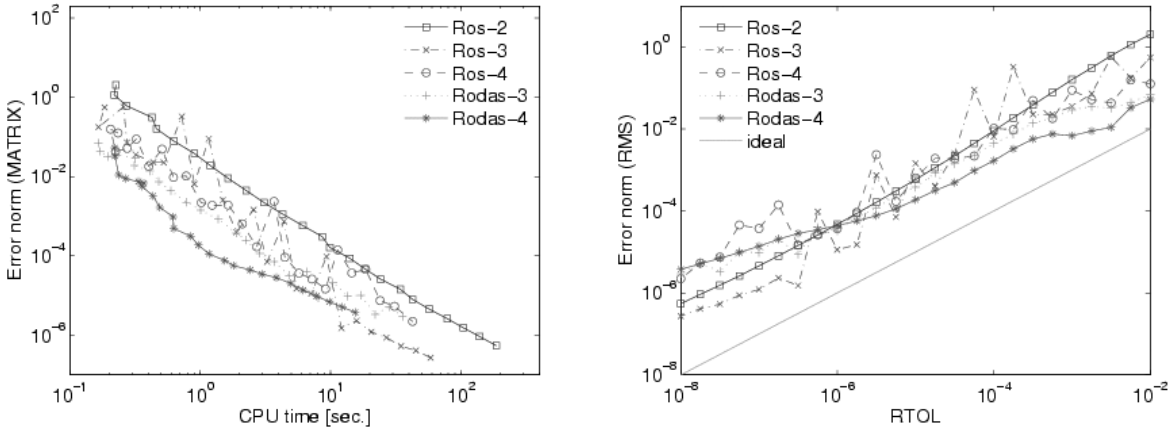

Figure 22: Accuracy vs. CPU time (left) and accuracy vs. tolerance (right) of adjoint Rosenbrock methods.

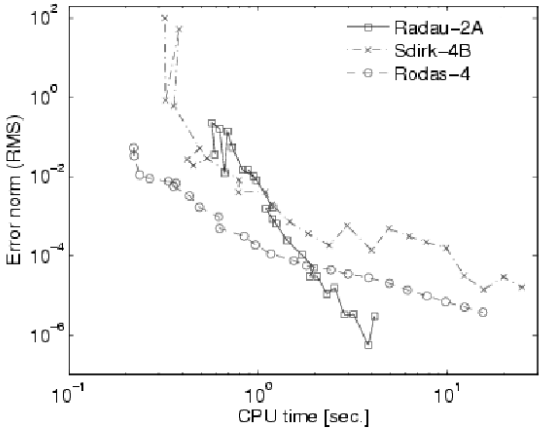

Figure 23: Accuracy vs. CPU-time for adjoint Radau-2A, adjoint Sdirk-4B, and adjoint Rodas-4 methods.

Variational methods are widely used in meteorological and oceanographic applications; more details can be found in [35].

We consider a 4D-VAR experiment with the SAPRC-99 chemical mechanism carried out in a twin experiment framework. A 48 hour reference run starting at $t_s = 12:00pm$ local time is considered as the "true solution" of the model. Artificial observations $\bar{y}_k$ are generated hourly from the reference solution by adding random Gaussian noise $\epsilon_k \in \mathcal{N}(0, R_k)$

$$\bar{y}_k = H_k y_k + \epsilon_k. \tag{30}$$

The noise simulates measurement errors. Only long-lived species are used as observations (and selected via the "observation operator" H_k).

The run is repeated with initial concentrations increased by 30%. ($y^B = 1.3y^{\text{ref}}$). These modified initial concentrations represent the "best guess" initial conditions. We look to recover the reference initial conditions using the information contained in the artificial observations. For this we define the following cost function:

$$\Psi(y^0) = \frac{1}{2}\left(y^B - y^0\right)^T B^{-1}\left(y^B - y^0\right) + \frac{1}{2}\sum_{k=0}^{m}\left(H_k y_k - \bar{y}_k\right)^T R_k^{-1}\left(H_k y_k - \bar{y}_k\right). \tag{31}$$

The background is a diagonal matrix with all diagonal entries equal to 0.01, while the observation covariances are diagonal matrices with all diagonal entries equal to 1.

This cost function measures the mismatch between the (perturbed) model solution and the (artificial) observations, and also penalizes the departure of the solution form the initial guess. The optimal initial state y^0 is obtained as the argument which minimizes the cost function. The minimization of (31) is carried out using the LBFGS-B [9] quasi-Newton optimization routine. For a better scaling and for imposing the positivity constraint the control variables are taken to be the logarithms of the initial concentrations ($\log y^0$). The gradient of (31) with respect to the control variables is obtained by solving the discrete adjoint model using the Lobatto-3C fully implicit Runge Kutta method.

The optimization results presented in Figure 24 are obtained after 51 L-BFGS iterations. Emission data is included but not varied over time. Tolerances are set to RTOL=10^{-4} and ATOL=10. The perturbed solutions are quite different than the reference solutions. After data assimilation the optimized

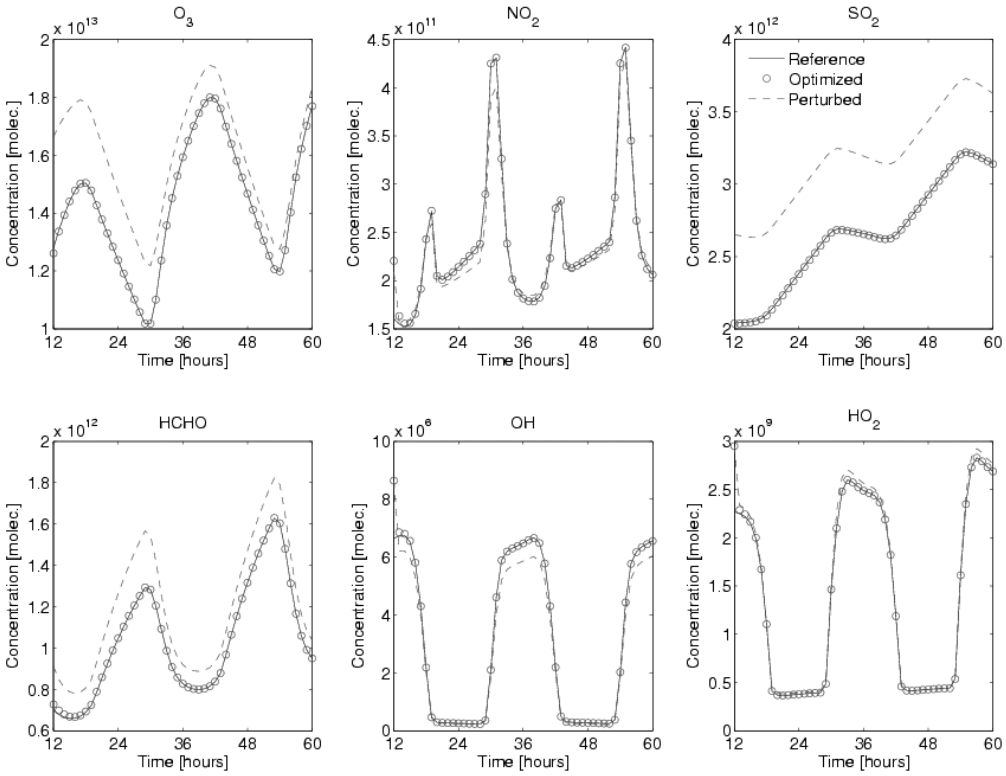

Figure 24: Time evolution of six species before and after data assimilation.

initial conditions lead to a solution that is indistinguishable from the reference one. Thus data assimilation, using the adjoint Lobatto-3C model, is successful in retrieving the reference initial conditions of the chemical model.

8 Code Availability

The KPP–2.2 source code is available for download under the Gnu Public License [3] from `http://www.cs.vt.edu/~asandu/Software/Kpp` [4]. The download archive contains the directory `kpp-2.2` with the entire KPP–2.2 source code. The following templates of the new integrators can be found under the directory `kpp-2.2/int`: `runge_kutta.f90`, `runge_kutta_tlm.f90`, `runge_kutta_adj.f90`, and `sdirk.f90`, `sdirk_tlm.f90`, `sdirk_adj.f90` respectively.

9 Conclusions

In this paper we report on state-of-the-art, high-order stiff Runge Kutta numerical integrators for efficient integration of chemical kinetic systems. Two families of implicit Runge Kutta methods have been implemented in the widely-used software environment KPP. One family is the fully implicit three-stage Runge Kutta methods (including Radau-1A, Radau-2A, Lobatto-3C and Gauss formulas). The second family are singly diagonally-implicit Runge Kutta (SDIRK) methods (including five different formulas of orders 2–4). Integrators in both these families have excellent stability properties and allow for efficient and high accuracy solutions of chemical kinetic systems.

Tangent linear versions of the fully implicit and singly diagonally implicit Runge Kutta methods are implemented in KPP–2.2 for direct decoupled sensitivity analysis. Implementation details specific to each family and the integration options available to the user are discussed. To our knowledge this work is the first to develop direct decoupled sensitivity analysis in the context of implicit Runge Kutta methods.

KPP–2.2 also offers new discrete adjoint implementations of the fully implicit and singly diagonally implicit Runge Kutta methods. Discrete adjoints offer a computationally efficient way to compute sensitivities of a cost functional with respect to the chemical model parameters, and are useful in the solution of inverse problems. The formulation of discrete Runge Kutta adjoints and implementation details specific to each family are discussed. To our knowledge the current work is the first publicly available software for discrete adjoints of fully and singly diagonally implicit Runge Kutta methods.

Comprehensive tests of the forward, tangent linear, and adjoint methods are performed with a chemical mechanism used in real air pollution applications. In addition we illustrate the use of discrete adjoints to solve a chemical kinetic inverse problem (4D-VAR data assimilation).

Only discrete adjoints are currently implemented in KPP–2.2 for fully implicit Runge Kutta and for SDIRK methods. Both discrete and continuous adjoints are implemented for the Rosenbrock methods. In the future we plan to implement continuous adjoints for both Runge Kutta families.

10 Acknowledgments

This work was supported by the National Science Foundation through the awards NSF ITR AP&IM 0205198, NSF CAREER ACI0413872, and NSF CCF0515170, by the National Oceanic and Atmospheric Administration (NOAA) and by the Texas Environmental Research Consortium (TERC).

A Coefficients of the SDIRK error estimators

In this appendix we provide the coefficients of the additional SDIRK stage for error estimation with the fully implicit methods. The embedded methods are characterized by $\hat{b}_i = \hat{a}_{5,i}$ (stiff accuracy).

The coefficients for Radau-2A:

$$\hat{b} = \begin{bmatrix} b_0 \\ \left(1/18 + 1/12\sqrt{6}\right)\left(-1 - 6\,b_0 + \sqrt{6}\right) \\ \left(-1/18 + 1/12\sqrt{6}\right)\left(6\,b_0 + 1 + \sqrt{6}\right) \\ -\frac{4}{45} - 1/3\,b_0 - 1/10\,\sqrt[3]{3} + 1/30\,3^{2/3} \\ 1/5 + 1/10\,\sqrt[3]{3} - 1/30\,3^{2/3} \end{bmatrix}$$

The coefficients for Lobatto-3C:

$$\hat{b} = \begin{bmatrix} b_0 \\ 1/6 - b_0 \\ 2/3 \\ 1/6 - \left(1/2\,\left(2 + 2\sqrt{3}\right)^{2/3} - \sqrt{3}\,\sqrt[3]{2 + \sqrt{3}} + \sqrt[3]{2 + 2\sqrt{3}} + 2\right)^{-1} \\ \left(1/2\,\left(2 + 2\sqrt{3}\right)^{2/3} - \sqrt{3}\,\sqrt[3]{2 + \sqrt{3}} + \sqrt[3]{2 + 2\sqrt{3}} + 2\right)^{-1} \end{bmatrix}$$

The coefficients for Gauss:

$$\hat{b} = \begin{bmatrix} 0 \\ 1/36\,\sqrt{3}\sqrt{5} + \frac{5}{36} + \frac{1}{144}\,2^{2/3}\sqrt[3]{\sqrt{5}+1}\sqrt{3} - \frac{1}{144}\,2^{2/3}\sqrt[3]{\sqrt{5}+1}\sqrt{3}\sqrt{5}\cdots \\ \cdots - \frac{1}{144}\,2^{2/3}\sqrt[3]{\sqrt{5}+1}\sqrt{5} + \frac{5}{144}\,2^{2/3}\sqrt[3]{\sqrt{5}+1} + \frac{1}{72}\,\sqrt[3]{2}\left(\sqrt{5}+1\right)^{2/3}\sqrt{3} - \frac{1}{72}\,\sqrt[3]{2}\left(\sqrt{5}+1\right)^{2/3}\sqrt{5} \\ \frac{1}{180}\,2^{2/3}\sqrt[3]{\sqrt{5}+1}\sqrt{5} - 1/36\,2^{2/3}\sqrt[3]{\sqrt{5}+1} + \frac{1}{90}\,\sqrt[3]{2}\left(\sqrt{5}+1\right)^{2/3}\sqrt{5} + 5/9 \\ \frac{5}{36} - 1/36\,\sqrt{3}\sqrt{5} - \frac{1}{144}\,\sqrt{3}\sqrt[3]{4+4\sqrt{5}} - \frac{1}{144}\,\sqrt{5}\sqrt[3]{4+4\sqrt{5}}\cdots \\ \cdots + \frac{5}{144}\,\sqrt[3]{4+4\sqrt{5}} + \frac{1}{144}\,\sqrt{3}\sqrt{5}\sqrt[3]{4+4\sqrt{5}}\cdots \\ \cdots - \frac{1}{144}\,\sqrt{5}\left(4+4\sqrt{5}\right)^{2/3} - \frac{1}{144}\,\sqrt{3}\left(4+4\sqrt{5}\right)^{2/3} \\ -1/24\,\sqrt[3]{4+4\sqrt{5}} + \frac{1}{120}\,\sqrt{5}\sqrt[3]{4+4\sqrt{5}} + 1/6 + \frac{1}{120}\,\sqrt{5}\left(4+4\sqrt{5}\right)^{2/3} \end{bmatrix}$$

The coefficients for Radau-1A:

$$\hat{b} = \begin{bmatrix} 0 \\ \frac{1}{90}\,3^{2/3} - 1/30\,\sqrt[3]{3} + \frac{2}{45} \\ -1/20\,\sqrt{2}\sqrt[6]{3} + \frac{23}{180}\,\sqrt{2}\sqrt{3} + 1/20\,\sqrt{2}3^{5/6} - 1/30\,\sqrt[3]{3} + \frac{1}{90}\,3^{2/3} + \frac{17}{45} \\ \frac{17}{45} - 1/30\,\sqrt[3]{3} + \frac{1}{90}\,3^{2/3} + 1/20\,\sqrt{2}\sqrt[6]{3} - \frac{23}{180}\,\sqrt{2}\sqrt{3} - 1/20\,\sqrt{2}3^{5/6} \\ 1/5 + 1/10\,\sqrt[3]{3} - 1/30\,3^{2/3} \end{bmatrix}$$

References

[1] AutoChem home page. `http://www.autochem.info`.

[2] CHEMKIN reactor models home page. `http://www.reactiondesign.com/products/open/chemkin.html`.

[3] GNU general public license. `http://www.gnu.org/copyleft/gpl.html`.

[4] The Kinetic PreProcessor home page. `http://www.cs.vt.edu/$\sim$asandu/Software/Kpp`.

[5] The Master Chemical Mechanism home page. `http://mcm.leeds.ac.uk/MCM`.

[6] MATLAB home page. `http://www.mathworks.com/products/matlab`.

[7] ODEPACK home page. `http://www.llnl.gov/CASC/odepack`.

[8] M.L. Baguer and W. Romisch. Computing gradients in parametrization-discretization schemes for constrained optimal control problems. Approximation and Optimization in the Carribbean II, p. 14–34, M. Florenzano editor, Peter Lang, Frankfurt am Main, 1995.

[9] R. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal of Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[10] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh. VODE: A variable step ODE solver. *SIAM Journal on Scientific and Statistical Computing*, 10:1038–1051, 1989.

[11] W.P.L. Carter. Documentation of the SAPRC-99 chemical mechanism for VOC reactivity assessment: Final report to California air resources board. California Air Resources Board Contract 92–329,95–308, California Air Resources Board, 2000.

[12] A. R. Curtis and W. P. Sweetenham. Facsimile/Chekmat user's manual. Technical report, Computer Science and Systems Division, Harwell Lab., Oxfordshire, Great Britain, August 1987.

[13] D. Daescu, A. Sandu, and G. R. Carmichael. Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: II – validation and numerical experiments. *Atmospheric Environment*, 37:5097–5114, 2003.

[14] V. Damian, A. Sandu, M. Damian, F. Potra, and G. R. Carmichael. The kinetic preprocessor KPP – a software environment for solving chemical kinetics. *Computers and Chemical Engineering*, 26:1567–1579, 2002.

[15] A. M. Dunker The decoupled direct method for calculating sensitivity coefficients in chemical kinetics. *Journal of Chemical Physics*, 81:2385–2402, 1984,

[16] R. Djouad, B. Sportisse, and N. Audiffren. Reduction of multiphase atmospheric chemistry. *Journal of Atmospheric Chemistry*, 46:131–157, 2003.

[17] M.B. Giles. On the Use of Runge-Kutta Time-Marching and Multigrid for the Solution of Steady Adjoint Equations. Technical Report NA00/10, Oxford University Computing Laboratory, 2000.

[18] K. Gustafsson. Control-theoretic techniques for stepsize selection in implicit Runge-Kutta methods. *ACM Transactions on Mathematical Software*, 20(4):496–517, 1994.

[19] W. Hager. Runge Kutta methods in optimal control and the transformed adjoint system. Numerische Mathematik 87(2):247–282, 2000.

[20] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer-Verlag, Berlin, 1993.

[21] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics, Berlin, 1996.

[22] J.R. Leis and M.A. Kramer. Algorithm 658: ODESSA – an ordinary differential equation solver with explicit simultaneous sensitivity analysis. *ACM Transactions on Mathematical Software*, 14(1):61–67, 1988.

[23] K. Radhakrishnan and A. Hindmarsh. *Description and use of LSODE, the Livermore solver for differential equations*. NASA reference publication 1327, 1993.

[24] R. Sander, A. Kerkweg, P. Jöckel, and J. Lelieveld. Technical note: The new comprehensive atmospheric chemistry module MECCA. *Atmospheric Chemistry and Physics*, 5:445–450, 2005.

[25] 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*, 31:3459–3472, 1997.

[26] A. Sandu, D. Daescu, and G. R. Carmichael. Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: I – Theory and software tools. *Atmospheric Environment*, 37:5083–5096, 2003.

[27] A. Sand, and R. Sander. Simulating Chemical Kinetic Systems in Fortran90 and Matlab with the Kinetic PreProcessor KPP-2.1. *Atmospheric Chemistry and Physics*, 6:187–195, SRef-ID: 1680-7324/acp/2006-6-187, 2006.

[28] A. Sandu and R. Sander: "KPP – User's Manual", `http://www.cs.vt.edu/$\sim$asandu/Software/Kpp`.

[29] P. Miehe and A. Sandu Forward, Tangent Linear, and Adjoint Runge Kutta Methods in KPP–2.2. V.N. Alexandrov et al. (Eds.): ICCS 2006, III, LNCS 3993, p. 120–127, Springer-Verlag Berlin Heidelberg, 2006.

[30] A. Sandu On the Properties of Runge Kutta Discrete Adjoints. V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, p. 550–557, Springer-Verlag Berlin Heidelberg, 2006.

[31] Y. Tang, G. R. Carmichael, I. Uno, J.-H. Woo, G. Kurata, B. Lefer, R. E. Shetter, H. Huang, B. E. Anderson, M. A. Avery, A. D. Clarke, and D. R. Blake. Impacts of aerosols and clouds on photolysis frequencies and photochemistry during TRACE-P: 2. three-dimensional study using a regional chemical transport model. *Journal of Geophysical Research*, 108D, 2003.

[32] J. Trentmann, M. O. Andreae, and H.-F. Graf. Chemical processes in a young biomass-burning plume. *Journal of Geophysical Research*, 108D, 2003.

[33] R. von Glasow, R. Sander, A. Bott, and P. J. Crutzen. Modeling halogen chemistry in the marine boundary layer. 1. Cloud-free mbl. *Journal of Geophysical Research*, 107D, 2002.

[34] R. von Kuhlmann, M. G. Lawrence, P. J. Crutzen, and P. J. Rasch. A model for studies of tropospheric ozone and nonmethane hydrocarbons: Model description and ozone results. *Journal of Geophysical Research*, 108D, 2003.

[35] K.Y. Wang, D.J. Lary, D.E. Shallcross, Hall S.M., and Pyle J.A. A review on the use of the adjoint method in four-dimensional atmospheric-chemistry data assimilation. *Quarterly Journal of Royal Meteorological Society*, 127(576 –Part B):2181–2204, 2001.

[36] A Walther. Automatic Differentiation of Explicit Runge Kutta methods for Optimal Control. Technical University Dresden technical report WR-06-2004. To appear in Journal of Computational Optimization and Applications.