
ELSEVIER

Journal of Computational and Applied Mathematics 88 (1997) 315–326

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

RKC: An explicit solver for parabolic PDEs

B.P. Sommeijer[a,*], L.F. Shampine[b], J.G. Verwer[a]

[a] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

[b] *Mathematics Department, Southern Methodist University, Dallas, TX 75275-0156, USA*

Received 3 June 1997; received in revised form 6 October 1997

Abstract

The FORTRAN program RKC is intended for the time integration of parabolic partial differential equations discretized by the method of lines. It is based on a family of Runge–Kutta–Chebyshev formulas with a stability bound that is quadratic in the number of stages. Remarkable properties of the family make it possible for the program to select at each step the most efficient stable formula as well as the most efficient step size. Moreover, they make it possible to evaluate the explicit formulas in just a few vectors of storage. These characteristics of the program make it especially attractive for problems in several spatial variables. RKC is compared to the BDF solver VODPK on two test problems in three spatial variables.

Keywords: Parabolic partial differential equations; Numerical software; Time integration; Runge–Kutta–Chebyshev solver

AMS classification: 65L05; 65M20; 65Y20

1. Introduction

RKC is a variable step size, variable formula code that uses explicit Runge–Kutta formulas to solve efficiently a class of large systems of mildly stiff ordinary differential equations (ODEs). The systems arising when a parabolic partial differential equation (PDE) is approximated by semi-discretization exemplify the problems for which RKC is intended. To be more specific, let the initial value problem for the ODEs have the form

$$\frac{dU(t)}{dt} = F(t, U(t)), \quad 0 < t \leqslant T, \quad U(0) = U_0, \tag{1.1}$$

* Corresponding author. E-mail: bsom@cwi.nl.

0377-0427/97/$17.00
PII S0377-0427(97)00219-7

so that the Jacobian matrix is $F'(t,U)=\partial F(t,U)/\partial U$. RKC is intended for problems with Jacobians that are close to normal and that have all their eigenvalues near the negative real axis. These properties are certainly true when $F'(t,U)$ is symmetric and nonpositive-definite, which is frequently the case when discretizing elliptic operators.

RKC exploits some remarkable properties of a family of explicit Runge–Kutta formulas of the Chebyshev-type proposed by van der Houwen and Sommeijer [18]. There is a member of s stages for all $s\geqslant 2$, and there are analytical expressions for its coefficients. All the formulas have stability regions that include narrow strips about the negative real axis. The length of the strip, the stability boundary $\beta(s)$, is approximated well by $0.653s^2$. This makes it possible for RKC to solve problems that are mildly stiff with explicit formulas. A very important property is that because of a recursion for Chebyshev polynomials, it is possible to evaluate a formula with just a few vectors of working storage, no matter the number of stages. Most remarkable is that for practical purposes the local errors of all members of the family are the same. This means that the code can estimate first the most efficient step size and then use an estimate of the spectral radius of the Jacobian to determine the most efficient formula for which this step size is stable. Another important property of the family is that it is easy to obtain a continuous extension of excellent quality that is "free". This is especially valuable for a Runge–Kutta formula that might involve a great many stages.

RKC has very modest storage requirements because it uses explicit formulas that can be evaluated by recursion. It requires at most seven vectors of storage. This makes it attractive for the solution of PDES in several space variables by semi-discretization. Another advantage of explicit formulas is that vectorization and/or parallelization presents no particular difficulties. The code is, for example, suitable for problems with solutions that are travelling waves because small steps are needed to resolve fronts accurately. Generally, reaction–diffusion systems

$$\frac{\partial u}{\partial t}=\nabla\cdot(K\,\nabla u)+f(u,x,t),\quad u=u(x,t),\ x\in\mathbb{R}^d,$$

where f is a modestly stiff reaction term can be solved efficiently with RKC. When f gives rise to severe stiffness, RKC is not recommended. In such cases it can still be useful as part of an operator splitting scheme that treats the reaction part at grid points with a standard code for stiff problems. Likewise, in combination with operator splitting RKC can be useful for systems of transport problems of advection–diffusion–reaction type

$$\frac{\partial u}{\partial t}+\nabla\cdot(a\,u)=\nabla\cdot(K\,\nabla u)+f(u,x,t),\quad u=u(x,t),\ x\in\mathbb{R}^d.$$

Problems of this kind play an important role in the modelling of pollution of the atmosphere, ground water, and surface water, and are the subject of much current research.

Section 2 presents the family of formulas implemented in RKC. The following section discusses the properties of the family that are crucial to the success of the solver and how they are exploited in software. Among the issues discussed are the estimation and control of error, estimation of the spectral radius and control of stability, and a continuous extension. Section 4 presents results for two PDES in three spatial variables taken from [12]. The last section explains how to obtain a copy of RKC and its auxiliary programs along with examples showing how to use them.

2. RKC's formulas

Historically, the principal goal when constructing Runge–Kutta formulas was to achieve the highest order possible with a given number of stages s. Stabilized methods are different, in that, the principal goal is to construct formulas with regions of absolute stability that are as large as possible in a sense that depends on the intended application. The formulas of RKC are intended for problems like those arising when parabolic PDEs are approximated by semi-discretization. Correspondingly, the goal is to construct formulas that are stable on a strip containing a long segment of the negative real axis. The wider the strip, the greater the applicability of the method, but the most important characteristic of the formula is the length of the segment, the stability boundary $\beta(s)$. For the ODEs of semi-discretization, a low-order formula is appropriate because only a modest accuracy is expected of the approximation to the PDE. When the PDE involves more than one spatial variable, the size of the system of ODEs grows rapidly as the mesh spacing is decreased. The relatively crude meshes that are used for this reason lead to relatively large discretization errors in space, hence limits the accuracy that would be meaningful in the time integration and so favors low-order methods. It turns out that the higher-order methods require more stages to achieve the same stability, another factor favoring low-order formulas. For these reasons all the formulas of RKC are of order two.

The formulas of RKC are given in [16]. To avoid confusion, we point out that they are slightly different from the formulas in [18]. A comprehensive linear stability and convergence analysis of the formulas is found in [20]. Amongst others, this analysis proves that the RKC formulas do not suffer from order reduction. The formulas are also studied in the review article [21] along with a number of related methods.

Let U_n denote the approximation to $U(t)$ at $t=t_n$ and let $\tau=t_{n+1}-t_n$ be the step size in the current step from t_n to t_{n+1}. The formulas of RKC have the form

$$\begin{aligned} &Y_0=U_n,\\ &Y_1=Y_0+\tilde{\mu}_1\tau F_0,\\ &Y_j=(1-\mu_j-\nu_j)Y_0+\mu_jY_{j-1}+\nu_jY_{j-2}+\tilde{\mu}_j\tau F_{j-1}+\tilde{\gamma}_j\tau F_0, \quad j=2,\ldots,s,\\ &U_{n+1}=Y_s. \end{aligned} \tag{2.1}$$

All the coefficients are available in analytical form for arbitrary $s\geqslant 2$. They are defined as follows. Let T_j be the Chebyshev polynomial of the first kind of degree j. Then

$$\varepsilon=\tfrac{2}{13}, \quad w_0=1+\varepsilon/s^2, \quad w_1=\frac{T_s'(w_0)}{T_s''(w_0)}, \quad b_j=\frac{T_j''(w_0)}{(T_j'(w_0))^2} \quad (2\leqslant j\leqslant s), \quad b_0=b_2, \quad b_1=b_2$$

and

$$\tilde{\mu}_1=b_1w_1, \quad \mu_j=\frac{2b_jw_0}{b_{j-1}}, \quad \nu_j=\frac{-b_j}{b_{j-2}}, \quad \tilde{\mu}_j=\frac{2b_jw_1}{b_{j-1}}, \quad \tilde{\gamma}_j=-(1-b_{j-1}T_{j-1}(w_0))\tilde{\mu}_j \quad (2\leqslant j\leqslant s).$$

In (2.1) the stage $F_j=F(t_n+c_j\tau,Y_j)$. The c_j are

$$c_j=\frac{T_s'(w_0)}{T_s''(w_0)}\frac{T_j''(w_0)}{T_j'(w_0)}\approx\frac{j^2-1}{s^2-1} \quad (2\leqslant j\leqslant s-1), \quad c_1=\frac{c_2}{T_2'(w_0)}\approx\frac{c_2}{4}, \quad c_s=1.$$

The approximations show that the arguments $t_n+c_j\tau$ all lie within the span of the step to $t_n+\tau$.

3. Software issues

RKC is the result of both software and algorithmic development of Sommeijer's code [17]. Broadly speaking, the implementation is like that of any modern code based on an explicit Runge–Kutta formula. In this section we describe briefly aspects of the code that are unusual or even unique. Any modern general-purpose code for initial value problems will estimate the local error at each step and adjust the step size both to control this error and to solve the problem efficiently. Popular Adams, BDF, and extrapolation codes also select the formula dynamically. The main difficulty in selecting the most efficient formula is in estimating the step size that could be used with a formula other than the one used to take the step. The family of formulas implemented in RKC has the remarkable property that for practical purposes, all the formulas have the same accuracy. The stability boundary of the formulas increases quadratically with the number of stages. By computing an estimate of the spectral radius, the code is able to determine the most efficient formula that is stable with a step size predicted to yield the desired accuracy. An important property of the family is that it is possible to evaluate a formula using just a few vectors of working storage, no matter how large the number of stages. Still another important property is that it is easy to obtain a continuous extension of excellent quality.

3.1. Error control

For a smooth F in (1.1), a Taylor series expansion of the local solution at $t=t_n$ results in

$$U_{n+1} = U + \tau \dot{U} + \tfrac{1}{2}\tau^2 \ddot{U} + C_{31,s}\tau^3 F_j F_k^j F^k + C_{32,s}\tau^3 F_{jk} F^j F^k + O(\tau^4), \quad s \geqslant 2.$$

Naturally, the coefficients $C_{31,s}$ and $C_{32,s}$ depend on the formula, i.e., on the number of stages s. However, it is found that both tend rapidly to a constant value as s increases. Indeed, they are both close to $\frac{1}{10}$ for all $s \geqslant 2$. This says that the leading term of the local error expansion is approximately proportional to the third derivative of the solution. As a consequence, the global error is approximately independent of s. The convergence results in [20] make this precise for linear problems. Extensive testing with both linear and nonlinear problems has confirmed that for practical purposes, the local error is independent of the number of stages for $s \geqslant 2$.

Let $Le(t_{n+1})$ be the approximation to the leading term of the local error expansion resulting from replacing the true s-dependent constants by their limiting values:

$$Le(t_{n+1}) = \tfrac{1}{15}\tau^3 \, \mathrm{d}^3 U(t_n)/\mathrm{d}t^3.$$

The simple form of this expression for the error makes it easy to obtain an asymptotically correct estimate:

$$Est_{n+1} = \tfrac{1}{15}\,[12(U_n - U_{n+1}) + 6\tau(F(U_n) + F(U_{n+1}))].$$

At each step the estimated local error is controlled so that accuracy tolerances specified by the user are met. There is a scalar relative error tolerance *rtol*. The user must ask for some relative accuracy, but not too much for the precision available. Because the formulas are of order two, the code is not appropriate for stringent tolerances. The absolute error tolerances can be supplied in the form of a scalar *atol* that is applied to all the solution components or as a vector that is applied

to corresponding components. A scalar absolute error tolerance is convenient and saves a useful amount of storage, but is appropriate only when all the solution components are on the same scale. These tolerances are used in the weighted RMS norm

$$\|\mathrm{Est}_{n+1}\| = \|w^{-1}\,\mathrm{Est}_{n+1}\|_2, \quad w = \sqrt{m}\,\mathrm{diag}(\mathrm{Tol}_1, \ldots, \mathrm{Tol}_m),$$

where

$$\mathrm{Tol}_k = \mathrm{atol}_k + \mathrm{rtol}|U_{n+1,k}|,$$

m is the dimension of the ODE system and $U_{n+1,k}$ the kth component of U_{n+1}. Hence, the step is accepted if $\|\mathrm{Est}_{n+1}\| \leqslant 1$ and otherwise rejected and redone. The error is controlled by an error per step criterion, so if all is going well, the arguments in [15] show that reducing the tolerances by a factor of 0.1 will reduce the error in the numerical solution by a factor of roughly 0.2.

A standard device for reducing the number of rejected steps is to use a fraction of the step size predicted to be optimal; a relatively small fraction is used in RKC. Watts [22] uses information gathered at the preceding step to refine the conventional prediction of the optimal step size. Later Gustafsson et al. [6] derived nearly the same algorithm from the completely different viewpoint of control theory. Versions of the algorithm are seen in RKSUITE [1] and RADAU5 [8]. These very successful codes have demonstrated the value of the refined prediction for reducing the number of step failures, so RKC also implements a version of the algorithm. Specifically, the prediction for the new step size after a successful step is given by

$$\tau_{\mathrm{new}} = \min(10, \max(0.1, \mathrm{fac}))\,\tau$$

with the fraction fac defined by

$$\mathrm{fac} = 0.8\left(\frac{\|\mathrm{Est}_n\|^{1/(p+1)}}{\|\mathrm{Est}_{n+1}\|^{1/(p+1)}}\,\frac{\tau_n}{\tau_{n-1}}\right)\frac{1}{\|\mathrm{Est}_{n+1}\|^{1/(p+1)}},$$

where p is the order of consistency, which is in our case equal to two. The conventional prediction is obtained by deleting the parenthesized term. It is used after a step rejection.

3.2. Initial step size

For the convenience of the user, RKC determines automatically an initial step size. In modern algorithms for this purpose, the main difficulty [5] is finding a step size that is on scale. Once this is done, a tentative step size can be refined by means of trial steps. The situation in RKC is special in two ways. A tentative step size τ_0 that is on scale is furnished by the reciprocal of the spectral radius that is computed for stability control. Further, the very simple form of the local error allows the error that would be made in a step of size τ to be estimated with a difference quotient [19] at a cost of a single function evaluation:

$$\tau_0 = 1/\sigma(F'(t_0, U_0)), \quad \mathrm{Est} = \tau_0(F(t_0 + \tau_0, U_0 + \tau_0 F(t_0, U_0)) - F(t_0, U_0)).$$

The initial step size τ_{start} is taken to be one-tenth of the largest step size predicted to satisfy the error test:

$$\tau_{\mathrm{start}} = 0.1\,\frac{\tau_0}{\|\mathrm{Est}\|^{1/2}}.$$

3.3. Absolute stability

At each step RKC first selects the "optimal" step size for controlling the local error and then selects a formula for which this step size is absolutely stable. Roughly speaking, the absolute stability regions of the formulas used are strips containing a segment of the negative real axis, cf. [21], and the length of the segment $\beta(s)$ is approximated well by $0.653s^2$. Assuming that the eigenvalues of local Jacobians lie in such a strip, the spectral radius of the Jacobian is all that is needed to find the smallest number of stages that yields stability for the step size τ:

$$\tau\sigma(F'(t,U)) \leqslant 0.653s^2. \tag{3.1}$$

Problems with constant Jacobians are sufficiently common that users are asked to identify them; RKC computes the spectral radius only once in such cases.

Sometimes it is easy enough to determine analytically a reasonably close upper bound on the spectral radius, using, e.g., Geršgorin's circle theorem, so RKC allows for this possibility. Generally, it is not expensive to evaluate such a bound, so the code invokes it at each successful step.

Commonly, RKC estimates the spectral radius automatically using a nonlinear power method. This is convenient for the user, but it does cost another vector of working storage and some computation. The basic idea of the power method is simple, but there are a good many ways the method can degenerate, so considerable care is needed in its implementation. Our implementation takes advantage of the experience reported in [10, 13, 14, 19], and here we describe only points that differ from previous work. An important difference is that it is assumed that the eigenvalues are close to the negative real axis. A Rayleigh quotient is then much more likely to reflect the magnitudes of the largest eigenvalues than in the general case of eigenvalues that might have substantial imaginary part. It is an upper bound on the spectral radius that is needed rather than the spectral radius itself, so the estimate is increased somewhat and it is then used conservatively in selecting the number of stages.

It is important to hold down the cost of computing the spectral radius. The slope of the solution at the beginning of a step (which is always available) is likely to be rich in the directions corresponding to dominant eigenvalues [13], so it is used to start the power method at the first step. We have found it very advantageous to retain the computed eigenvector from one estimation of the spectral radius for use as the starting guess for the next. With such a good guess it is typical that only a few iterations are needed. Still, the Jacobian should change slowly, so it should not be necessary to estimate the spectral radius at every step. The spectral radius is estimated on a step failure because this may indicate a change in the character of the problem. Otherwise, it is estimated for every 25 successful steps since the last estimate. Of course, unnecessary estimates are avoided when there are repeated step failures.

3.4. Storage

The form (2.1) for the formulas of RKC results from the three-term recursion relation for Chebyshev polynomials. It could be rewritten in the standard form of an explicit Runge–Kutta formula of s stages, but (2.1) is much better for computation. One reason will be taken up shortly, but the most important reason is that it is obvious this form of the formula can be evaluated using just a few vectors of working storage, no matter how large the number of stages. The precise amount of storage required by RKC depends on how the code is used, but it never uses more than five vectors

of storage for the computation itself. This makes it possible for RKC to solve the very large systems of ODEs arising from semi-discretization of PDEs in several spatial variables.

3.5. Internal stability

For conventional explicit Runge–Kutta methods, the accumulation of roundoff in the course of a step is unimportant, but that is not the case for methods with a large number of stages. Indeed, in the application of stabilized methods to parabolic PDEs, there can be a serious accumulation of rounding error, so serious that the number of stages must be limited [19–21]. The form (2.1) minimizes this internal instability, but there is still potential for a growth of roundoff at a modest rate proportional to s^2. For the problems that are the object of RKC and a reasonable working precision, such a growth presents no difficulties. However, for robustness the number of stages is limited in RKC to prevent an unacceptable growth of roundoff in the course of a step. According to [20], a safe assumption about this growth is that it is bounded by a relative perturbation of $10s^2\,uround$, where *uround* is the unit roundoff. The design of RKC emphasizes relative error, so it is required that this perturbation be no greater than rtol. Should the code find that it needs to use a larger s for stability with the desired step size, the number of stages is limited and the absolute stability condition (3.1) is satisfied by reducing the step size.

3.6. Continuous extension

Early codes based on explicit Runge–Kutta methods provide answers at specific points by shortening the step size. This is inefficient, especially when the method has many stages like those of RKC, so modern codes make use of a continuous extension to obtain cheaply answers anywhere in the span of a step. Cubic Hermite interpolation to the value and slope at the two ends of a step proves very satisfactory in the circumstances. It is easy to implement and provides a globally C^1 piecewise-polynomial solution. The interpolant is "free" because the slopes are computed for other purposes. It is shown in [4] that to leading order, the error of this interpolant is independent of the problem. Further, the error increases smoothly from the beginning of the step to a maximum at the end of the step. The error at the end of the step is the local error controlled by the code. Accordingly, to leading order the C^1 piecewise-polynomial solution is uniformly as accurate as the values at the mesh points. RKC is organized so that it can return after each step with the step size taken and all the information required for interpolation stored in a work array. The interpolant is evaluated at a point within the span of the step by calling an auxiliary subroutine RKCINT with the point and the work array as arguments.

4. Numerical examples

In this section we present numerical results for two examples considered in [12]. Both are parabolic PDEs in three space dimensions. Moore and Dillon use high-order finite elements for the spatial discretization and integrate the ODEs with DASPK. DASPK is a variant of DASSL that uses Krylov methods to make practical the evaluation of the implicit BDFs for "large" systems of ODEs and DAEs [3]. Because our main purpose here is to illustrate the use of RKC, we have discretized the PDEs with

central differences on a uniform grid. Although the techniques in [12] are very different, solving the same examples provides some perspective about the use of explicit methods for such problems. We include results computed with VODPK, a BDF code similar to DASPK. It is a modification of VODE [2] and is available from netlib: send vodpk.f from ode. It uses a preconditioned Krylov method GMRES for the solution of the linear systems with matrix $A = I - h\gamma F'$, where F' is the Jacobian. Since iterative methods such as GMRES require only matrix-vector products, A itself need not be stored, reducing greatly the memory needed in the solution of three-dimensional PDES. VODPK asks the user to specify the preconditioner P. For simplicity, in our experiments we used diagonal preconditioning, i.e., $P = I - h\gamma \operatorname{diag}(F')$. With this choice, the convergence behavior of GMRES is reasonable and the storage requirement of 19 vectors is acceptable. In contrast, RKC requires only five vectors for the first example and six for the second. Default values were used for all the parameters of VODPK. All computations were performed in double precision ($\approx$16 digits) on an SGI workstation with a 180 MHZ MIPS R5000 processor.

Example 1. The first example is the linear heat conduction problem

$$u_t = \Delta u + f(x, y, z, t), \quad 0 < x, y, z < 1, \quad t > 0,$$

where f, $u(x, y, z, 0)$, and Dirichlet boundary conditions are specified so that the solution is $u(x, y, z, t) = \tanh(5(x + 2y + 1.5z - 0.5 - t))$. The problem is solved for $0 \leqslant t \leqslant 0.7$. A uniform grid with spacing $h = 0.025$ is used, corresponding to $39^3 = 59\,319$ equations.

An analytical bound for the spectral radius of the Jacobian can be found easily by applying Geršgorin's circle theorem to the discrete Laplacian. Because three-point central differences are used, all rows of the matrix corresponding to an interior grid point have the form $h^{-2}(\ldots 1 \ldots 1 \ldots 1 \; -6 \; 1 \ldots 1 \ldots 1 \ldots)$, where "..." represents zero entries. For these rows the circle theorem yields a bound of $12/h^2$. Rows corresponding to a boundary point have more zero entries because of the Dirichlet boundary conditions. Thus, $12/h^2$ is an upper bound for the spectral radius and it turns out that the true radius is only marginally smaller. For $h = 0.025$, $\sigma \approx 19\,200$, so this problem is rather stiff for RKC.

Results reported here were computed with scalar tolerances rtol = atol = tol. For a range of tol, Table 1 presents the following quantities: the integration error at the end of the integration measured in the maximum norm, the total number of steps with the number of rejected ones parenthesized, the total number of F-evaluations, the average number of F-evaluations per step (both accepted and rejected), and the CPU time on the workstation in seconds. The error displayed in the table is the time integration error, being the difference between the fully discrete numerical solution and a reference solution of the ODES computed with a stringent tolerance. It would have been easier to compare the numerical solution to the analytical solution of the PDE, but this would be misleading because it mixes the error of the spatial discretization of the PDES (being $0.36 \cdot 10^{-2}$ in the maximum norm) with the error made in the time integration of the ODES. For the same tolerances rtol = atol = tol, Table 2 presents results for VODPK.

We see that both RKC and VODPK successfully solve the problem for all the tolerances, but RKC is better at delivering an accuracy comparable to the tolerance. The behavior of VODPK is particularly unsatisfactory when tol is reduced from 10^{-5} to 10^{-6}. The efficiency of the solvers is compared in Fig. 1 where the CPU time is plotted against the accuracy achieved. RKC is seen to compete well over the whole range of tolerances.

Table 1
Results for RKC for Example 1

Tol	Error	# Steps	# F-evals	Average #	CPU
10^{-1}	$0.89\cdot10^{-2}$	6(1)	402	67.0	186
10^{-2}	$0.17\cdot10^{-2}$	15(4)	729	48.6	338
10^{-3}	$0.37\cdot10^{-3}$	27(2)	786	29.1	366
10^{-4}	$0.39\cdot10^{-4}$	57(0)	1087	19.1	507
10^{-5}	$0.43\cdot10^{-5}$	129(1)	1682	13.0	787
10^{-6}	$0.65\cdot10^{-6}$	262(0)	2445	9.3	1149

Table 2
Results for VODPK for Example 1

Tol	Error	# Steps	# F-evals	CPU
10^{-1}	0.99	7(0)	46	35
10^{-2}	$0.83\cdot10^{-1}$	16(0)	160	122
10^{-3}	$0.10\cdot10^{-1}$	34(0)	237	185
10^{-4}	$0.12\cdot10^{-2}$	70(0)	474	371
10^{-5}	$0.13\cdot10^{-4}$	112(3)	984	770
10^{-6}	$0.19\cdot10^{-4}$	168(1)	1151	913

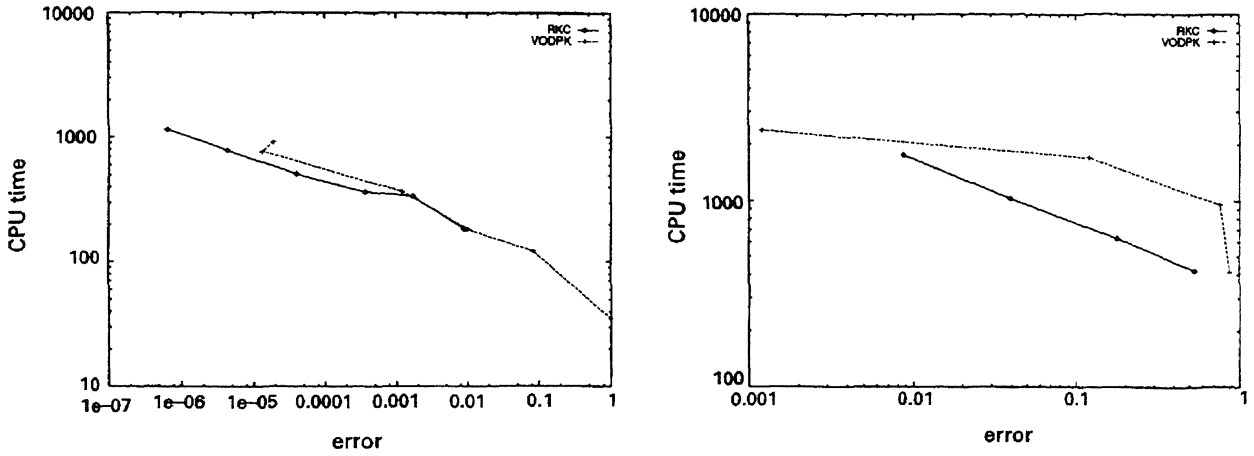

Fig. 1. A log–log plot of CPU time versus error for Examples 1 (left) and 2 (right) for RKC and VODPK.

Example 2. This example is a combustion problem described by the PDES:

$$c_t = \Delta c - Dce^{-\delta/T}, \quad LT_t = \Delta T + \alpha Dce^{-\delta/T}, \quad 0<x,y,z<1, \quad t>0,$$

along with the initial condition $c(x,y,z,0)=T(x,y,z,0)=1$, homogeneous Neumann boundary conditions for $x=y=z=0$ and the Dirichlet conditions $c(x,y,z,t)=T(x,y,z,t)=1$ for $x=y=z=1$. The parameters of the problem are $L=0.9$, $\alpha=1$, $\delta=20$ and $D=Re^{\delta}/\alpha\delta$ with $R=5$. The dependent

Table 3
Results for RKC for Example 2

Tol	Error	# Steps	# F-evals	Average #	# F-evals σ	CPU
10^{-4}	0.54	51 (1)	525	10.3	21	420
10^{-5}	0.18	124 (0)	781	6.3	27	630
10^{-6}	$0.39\cdot10^{-1}$	270 (0)	1270	4.7	39	1030
10^{-7}	$0.87\cdot10^{-2}$	581 (0)	2147	3.7	65	1758

Table 4
Results for VODPK for Example 2

Tol	Error	# Steps	# F-evals	CPU
10^{-4}	0.87	33 (2)	285	412
10^{-5}	0.76	91 (8)	659	957
10^{-6}	0.12	201 (9)	1141	1702
10^{-7}	$0.12\cdot10^{-2}$	286 (10)	1548	2376

variables c and T are the concentration and temperature of a chemical that is undergoing a one-step reaction. The temperature distribution develops a so-called "hot spot" at the origin. Ignition occurs at a finite time and T increases sharply to about $1+\alpha$. A reaction front is formed that propagates towards the boundary planes $x=y=z=1$ where it develops a boundary layer and finally ends in a steady state. Following [12] we solve the problem for $0\leqslant t\leqslant 0.3$. By the end of this period the boundary layers have developed and the solution is approaching steady state. A uniform grid with spacing h was used and the Neumann boundary conditions were discretized by means of central differences with fictitious points outside the region at a distance of $\frac{1}{2}h$. The grid spacing $h=1/(N+0.5)$ where N is the number of grid points in each of the three spatial variables. In the computations reported here $N=40$, leading to a total of $2\times 40^3=128\,000$ equations.

RKC is a natural candidate for the numerical integration of this flame propagation problem. For one thing, the travelling reaction front limits the step size of any integration scheme, be it implicit or explicit. For another, the problem becomes locally unstable in the course of the integration [21], so rather small steps are required to obtain an accurate solution in the transient phase, especially during ignition. Only during the start and near steady state it is possible to increase the step size to the point that an implicit method is competitive.

Tables 3 and 4 present results in the same way as for the first example. An extra column in Table 3 shows the number of F-evals needed by RKC for the estimation of the spectral radius. We see that the overhead for this automatic estimation is negligible. Both solvers integrate this difficult problem successfully with only a few step rejections. Neither code obtains accuracies comparable to the tolerance, though again RKC is notably better. With VODPK there is a striking change in accuracy when reducing tol from 10^{-6} to 10^{-7}. The low-accuracy achieved by both codes is to be expected from the local instability of the problem. Fig. 1 shows that RKC competes well with VODPK for this problem, too. RKC adapts the formula, i.e., the number of stages s, to the problem and it

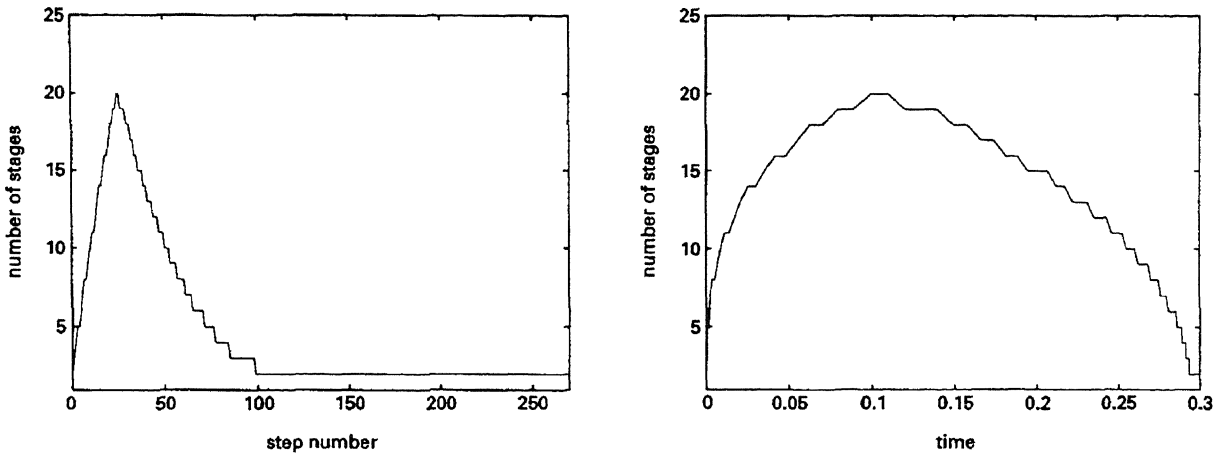

Fig. 2. The number of stages s used by RKC when solving Example 2 with $\mathrm{tol} = 10^{-6}$ plotted against step number (left) and against time (right).

may use s that are quite large compared to what is seen in general-purpose codes based on explicit Runge–Kutta formulas. The variation of s when solving this problem is displayed in Fig. 2.

5. Remarks

Other interesting stabilized explicit methods have been developed by Lebedev and co-workers, see, e.g., [9, 11, 8, 21]. There are formulas of order up to four [11]. Although they are also based on Chebyshev polynomials and so possess optimal stability for real negative eigenvalues, the three-term recursion is not exploited. A code DUMKA based on these formulas is still in an experimental stage, but numerical results are promising, see [8, Fig. 10.14].

Source code for RKC and some examples can be obtained by anonymous ftp from the address ftp://ftp.cwi.nl/pub/bsom/rkc. RKC can also be downloaded from netlib@ornl.gov (send rkc.f from ode). It replaces the program in [17].

References

[1] R.W. Brankin, I. Gladwell, L.F. Shampine, RKSUITE: a suite of Runge–Kutta codes for the initial value problem for ODEs, Softreport 91-1, Math. Dept., Southern Methodist Univ., Dallas, 1991.

[2] P.N. Brown, G.D. Byrne, A.C. Hindmarsh, VODE, a variable-coefficient ODE solver, SIAM J. Sci. Statist. Comput. 10 (1989) 1038–1051.

[3] P.N. Brown, A.C. Hindmarsh, L.R. Petzold, Using Krylov methods in the solution of large-scale differential-algebraic systems, SIAM J. Sci. Comput. 15 (1994) 1467–1488.

[4] I. Gladwell, L.F. Shampine, L.S. Baca, R.W. Brankin, Practical aspects of interpolation in Runge–Kutta codes, SIAM J. Sci. Statist. Comput. 8 (1987) 322–341.

[5] I. Gladwell, L.F. Shampine, R.W. Brankin, Automatic selection of the initial step size for an ODE solver, J. Comput. Appl. Math. 18 (1987) 175–192.

[6] K. Gustafsson, M. Lundh, G. Söderlind, A PI stepsize control for the numerical solution of ordinary differential equations, BIT 28 (1988) 270–287.

[7] E. Hairer, S.P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I, Nonstiff Problems, Springer, Berlin, 1987.
[8] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems, 2nd ed., Springer, Berlin, 1996.
[9] V.I. Lebedev, How to solve stiff systems of differential equations by explicit methods, in: G.I. Marchuk (Ed.), Numerical Methods and Applications, CRC Press, Boca Raton, FL, 1994, pp. 45–80.
[10] B. Lindberg, IMPEX: a program package for solution of systems of stiff differential equations, Report NA 72.50, Royal Inst. Technology, Stockholm, 1972.
[11] A.A. Medovikov, High order explicit methods for stiff ordinary differential equations, preprint, Russian Academy of Sciences, 1996.
[12] P.K. Moore, R.H. Dillon, A comparison of preconditioners in the solution of parabolic systems in three space dimensions using DASPK and a high order finite element method, Appl. Numer. Math. 20 (1996) 117–128.
[13] L.F. Shampine, Lipschitz constants and robust ODE codes, in: J.T. Oden (Ed.), Computational Methods in Nonlinear Mechanics, North-Holland, New York, 1980, pp. 427–449.
[14] L.F. Shampine, Diagnosing stiffness for Runge–Kutta methods, SIAM J. Sci. Statist. Comput. 12 (1991) 260–272.
[15] L.F. Shampine, Numerical Solution of Ordinary Differential Equations, Chapman & Hall, New York, 1994.
[16] B.P. Sommeijer, J.G. Verwer, A performance evaluation of a class of Runge–Kutta–Chebyshev methods for solving semi-discrete parabolic differential equations, Report NW91/80, Mathematisch Centrum, Amsterdam, 1980.
[17] B.P. Sommeijer, RKC, a nearly-stiff ODE solver, Available from netlib@ornl.gov, send rkc.f from ode, 1991.
[18] P.J. van der Houwen, B.P. Sommeijer, On the internal stability of explicit m-stage Runge–Kutta methods for large values of m, ZAMM 60 (1980) 479–485.
[19] J.G. Verwer, An implementation of a class of stabilized explicit methods for the time integration of parabolic equations, ACM Trans. Math. Software 6 (1980) 188–205.
[20] J.G. Verwer, W.H. Hundsdorfer, B.P. Sommeijer, Convergence properties of the Runge–Kutta–Chebyshev method, Numer. Math. 57 (1990) 157–178.
[21] J.G. Verwer, Explicit Runge–Kutta methods for parabolic partial differential equations, Appl. Numer. Math. 22 (1996) 359–379.
[22] H.A. Watts, Step size control in ordinary differential equation solvers, Trans. Soc. Comput. Simulation 1 (1984) 15–25.