Modelling, Analysis and Simulation

IRKC: an IMEX solver for stiff diffusion-reaction PDEs

L.F. Shampine, B.P. Sommeijer, J.G. Verwer

Report MAS-E0513 July 2005

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

IRKC: an IMEX solver for stiff diffusion-reaction PDEs

ABSTRACT

The Fortran 90 program IRKC is intended for the time integration of partial differential equations (PDEs) of diffusion-reaction type for which the reaction Jacobian has real (negative) eigenvalues. It is based on a family of implicit-explicit Runge-Kutta-Chebyshev formulas which are unconditionally stable for reaction terms and which provide a stability bound for the diffusion terms that is quadratic in the number of stages. Special 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 formulas in just a few vectors of storage. These characteristics of the program make it especially attractive for problems in several spatial variables. IRKC is a successor to the RKC code of [5] that solves similar problems without stiff reaction terms.

2000 Mathematics Subject Classification: 65M20; 65L05; 65Y99.
1998 ACM Computing Classification System: G.1.7; G.1.8; G4.
Keywords and Phrases: Numerical software; diffusion-reaction PDEs; time integration.
Note: CWI work carried out under subtheme MAS1.3 - Numerical Analysis of PDEs.

IRKC: an IMEX Solver for Stiff Diffusion-Reaction PDEs

L.F. Shampine
Mathematics Department, Southern Methodist University
Dallas, TX 75275-0156, USA

B.P. Sommeijer* & J.G. Verwer
Center for Mathematics and Computer Science (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

July 13, 2005

Abstract

The Fortran 90 program IRKC is intended for the time integration of partial differential equations (PDEs) of diffusion-reaction type for which the reaction Jacobian has real (negative) eigenvalues. It is based on a family of implicit-explicit Runge–Kutta–Chebyshev formulas which are unconditionally stable for reaction terms and which provide a stability bound for the diffusion terms that is quadratic in the number of stages. Special 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 formulas in just a few vectors of storage. These characteristics of the program make it especially attractive for problems in several spatial variables. IRKC is a successor to the RKC code of [5] that solves similar problems without stiff reaction terms.

2000 Mathematics Subject Classification: Primary: 65M20, 65L05, 65Y99.
1998 ACM Computing Classification System: G.1.7, G.1.8 and G4.
Keywords and Phrases: Numerical software, diffusion-reaction PDEs, time integration.
Note: CWI work carried out under subtheme MAS1.3 - Numerical Analysis of PDEs.

1 Introduction

The RKC code of [5] solves a system of NEQN first order ordinary differential equations (ODEs) of the form

$$y'(t) = F_E(t, y)\,. \tag{1}$$

It is intended for problems arising in the semi-discretization of parabolic partial differential equations (PDEs) in one or more space variables, but it accepts problems of relatively general form. Basic to this solver is the assumption that the eigenvalues of the Jacobian of F_E are on, or close to, the negative real axis. The solver implements a family of explicit, second-order Runge–Kutta–Chebyshev methods. At each step it selects a method that is both stable and efficient. The approach is attractive when the problem is at most moderately stiff and NEQN is relatively large. The paper [6] considers how to solve equations of the form

$$y'(t) = F_E(t, y) + F_I(t, y)\,, \tag{2}$$

[*] Corresponding author. *Email address*: B.P.Sommeijer@cwi.nl.

when the reactions described by the F_I term cause the ODEs to be very stiff and the reaction Jacobian F_I' possesses a real spectrum. In the IRKC code that we present here, the term F_E is handled by a variant of the explicit methods of RKC and the term F_I is handled implicitly. Such schemes are known as IMEX methods. Implicit treatment of large systems of ODEs has serious implications for storage, so we make assumptions about the coding of F_I that allow us to solve a useful class of problems with minimal storage. We have in mind systems of ODEs that arise from semi-discretization of diffusion-reaction PDEs. We make no assumption about how the discretization is done, but we do assume that the implicit terms at one grid point are not coupled to those at other grid points. This kind of decoupling arises naturally with finite differences, finite volumes, and discontinuous Galerkin discretizations. To take advantage of the great reductions in storage possible because of this assumption and the numerical method employed, the user must present the problem to the solver in an appropriate way. If there are NPDES partial differential equations, each call to the subroutine that evaluates F_I is made with a vector y of NPDES components that correspond to a single grid point. On demand the subroutine is also to evaluate the Jacobian of F_I, a matrix that is only NPDES by NPDES.

The papers [5, 6] lay the foundation for the IMEX formulas studied here, but there are some differences between those formulas and the ones implemented in IRKC. We derive here and implement in IRKC a measure of error that is different from that of the papers cited. The software we present could be described as an extension of the Fortran 77 code RKC of [5], but it is a considerable extension because we have redesigned the interface and exploited features of Fortran 90 to make it easier to use and at the same time solve problems that are more general.

In Section 2 we review briefly the IMEX formulas of [6] and derive the error assessment used in selecting the step size. Section 3 is devoted to software issues and their resolution in Fortran 90. Section 4 contains numerical examples of the use of IRKC.

2 The IMEX RKC Family of Formulas

Let y_n, y_{n+1} denote approximations to $y(t_n), y(t_{n+1})$, respectively, with step size $\tau_n = t_{n+1} - t_n$. For $s \geq 2$ stages, the IMEX family of formulas of [6] is defined by

$$\begin{aligned}
Y_0 &= y_n\,,\\
Y_1 &= Y_0 + \tilde{\mu}_1 \tau_n F_{E,0} + \tilde{\mu}_1 \tau_n F_{I,1}\,,\\
Y_j &= (1-\mu_j-\nu_j)\,Y_0 + \mu_j Y_{j-1} + \nu_j Y_{j-2} + \tilde{\mu}_j \tau_n F_{E,j-1} + \tilde{\gamma}_j \tau_n F_{E,0} + \\
&\qquad [\tilde{\gamma}_j - (1-\mu_j-\nu_j)\,\tilde{\mu}_1]\tau_n F_{I,0} - \nu_j \tilde{\mu}_1 \tau_n F_{I,j-2} + \tilde{\mu}_1 \tau_n F_{I,j}\,,\\
y_{n+1} &= Y_s\,,
\end{aligned} \tag{3}$$

where $j = 2, \ldots, s$ and $F_{E,j} = F_E(t_n + c_j \tau_n, Y_j)$, $F_{I,j} = F_I(t_n + c_j \tau_n, Y_j)$. All the coefficients are available in analytical form:

$$\begin{aligned}
\tilde{\mu}_1 &= b_1 \omega_1 \quad \text{and for} \quad j = 2, \ldots, s\,,\\
\mu_j &= \frac{2 b_j \omega_0}{b_{j-1}}\,, \quad \nu_j = \frac{-b_j}{b_{j-2}}\,, \quad \tilde{\mu}_j = \frac{2 b_j \omega_1}{b_{j-1}}\,, \quad \tilde{\gamma}_j = -(1-b_{j-1}) T_{j-1}(\omega_0) \tilde{\mu}_j\,,
\end{aligned} \tag{4}$$

where

$$b_0 = 1/(4\omega_0^2)\,, \quad b_1 = 1/\omega_0\,, \quad b_j = T_j''(\omega_0)\,/\,(T_j'(\omega_0))^2\,, \qquad j = 2, \ldots, s\,, \tag{5}$$

with $\omega_0 = 1 + \epsilon/s^2$, $\omega_1 = T_s'(\omega_0)/T_s''(\omega_0)$. Here T_k is the Chebyshev polynomial of the first kind and degree k. For arguments $\omega_0 > 1$, it has the form

$$T_k(\omega_0) = \cosh(k \operatorname{arccosh}(\omega_0)) = \cosh(k\,\ln(\omega_0 + \sqrt{\omega_0^2 - 1}))\,.$$

Crucial to the good performance of the method is that it has been developed from the Chebyshev recursion

$$T_0(z) = 1,\;\; T_1(z) = z,\;\; T_j(z) = 2zT_{j-1}(z) - T_{j-2}(z),\;\; 2 \le j \le s, \tag{6}$$

which holds for any $z \in \mathbb{C}$. The increments c_j are

$$\begin{aligned} c_0 &= 0, \quad c_1 = \omega_1/\omega_0, \quad \text{and for} \quad j = 2, \dots, s, \\ c_j &= T_s'(\omega_0)T_j''(\omega_0)/(T_s''(\omega_0)T_j'(\omega_0)) = \mu_j c_{j-1} + \nu_j c_{j-2} + \tilde{\mu}_j + \tilde{\gamma}_j. \end{aligned}$$

The parameter $\epsilon \ge 0$ is free and we use it to create damping. As in RKC [5] we set $\epsilon = 2/13$. The real stability boundary $\beta_E(s)$ corresponding to the part of the ODEs treated explicitly, i.e., the boundary when F_I is not present, is given by (see [6])

$$\beta_E(s) \;\approx\; \frac{(\omega_0+1)T_s''(\omega_0)}{T_s'(\omega_0)} \;\approx\; \tfrac{2}{3}\,(s^2-1)(1-\tfrac{2}{15}\epsilon) \approx 0.653\,(s^2-1)\,. \tag{7}$$

The number of stages s in the scheme is determined on the basis of a stable treatment of the F_E term. Given a step size τ_n (chosen as in Section 2.4) and (an upper bound on) the spectral radius ρ of the Jacobian F_E', the number of stages is set to the smallest value of s that satisfies the linear stability condition

$$\tau_n\,\rho(F_E') \le \beta_E(s) \;\approx 0.653\,(s^2-1)\,.$$

Remark. In many applications described by diffusion-reaction PDEs, scientists are interested in transient behavior and equilibria or steady-state solutions y for autonomous problems, $F_E(y) + F_I(y) = 0$. Standard Runge–Kutta and linear multistep methods preserve steady states. This property is shared by all the stages of (3) and the result of the step itself. In contrast, when operator splitting decouples the integration of the subsystems $y' = F_E(y)$ and $y' = F_I(y)$ within time steps (time splitting), equilibria are not preserved. For nonlinear problems with strongly competing subsystems F_E and F_I, preservation of equilibria is of obvious importance.

2.1 Solving the Implicit Relations

As in RKC, the diffusion operator F_E in the IMEX scheme is treated explicitly, exploiting the underlying three-term Chebyshev recursion (6) for stabilization. The difference now is that (3) is implicit in the stiff reaction term F_I, requiring at each stage the solution of a system of nonlinear algebraic equations

$$Y_j - \tilde{\mu}_1\tau_n F_I(t_n + c_j\tau_n, Y_j) = V_j\,, \tag{8}$$

with V_j given and Y_j an unknown vector of dimension NEQN. Because F_I has no underlying spatial grid connectivity, this system consists of a large number (the number of grid points) of uncoupled systems with dimension NPDES, the number of PDEs. As is customary when integrating stiff ODEs, we solve these systems by a modified Newton method. For each subsystem, i.e., for each stage and each grid point, the Jacobian of F_I and an LU decomposition of an iteration matrix are formed. This is expensive, but it allows a dramatic reduction in storage. A modified Newton process is used, meaning that for each subsystem, one iteration matrix is decomposed and this decomposition is used for all the iterations needed to solve the subsystem.

2.2 Local Truncation Error

Following standard practice for ODE solvers [4], the adjustment of step size is based on a study of the local truncation error. For this purpose we consider first the stability function obtained by applying (3) to the scalar test equation

$$y' = (\lambda_E + \lambda_I)y\,, \tag{9}$$

where λ_E and λ_I represent diffusion and reaction eigenvalues, respectively. The stability function is given in [6] as

$$R_s(z_E, z_I) = 1 - b_s T_s(\omega_0) + b_s T_s(\omega_0 + \omega_1 \tilde{z})\,, \tag{10}$$

where

$$z_E = \tau_n \lambda_E, \quad z_I = \tau_n \lambda_I, \quad z = z_E + z_I, \quad \tilde{\mu}_1 = \omega_1/\omega_0, \quad \tilde{z} = \frac{z}{1 - \tilde{\mu}_1 z_I}\,.$$

It is shown in [6] that $|R_s(z_E, z_I)| \leq 1$ for all pairs (z_E, z_I) satisfying

$$-0.653\;(s^2-1) \approx -\beta_E(s) \leq z_E \leq 0 \quad \text{and} \quad z_I \leq 0.$$

The stability function is an approximation to $\exp(\tilde{z})$ and $\exp(z)$. Expanding it for small z, we have

$$\begin{aligned} R_s(z_E, z_I) &= 1 + \frac{z}{1-\tilde{\mu}_1 z_I} + \frac{1}{2}\left(\frac{z}{1-\tilde{\mu}_1 z_I}\right)^2 + \mathcal{O}(z^3) \\ &= 1 + z + \frac{1}{2}z^2 + \tilde{\mu}_1 z z_I + \mathcal{O}(z^3, z z_I^2, z^2 z_I)\,. \end{aligned} \tag{11}$$

Comparing this expansion to that of the exponential, we see that the consistency order of 2 of the explicit method is reduced to 1 because of the reaction term that is treated implicitly. However, the coefficient $\tilde{\mu}_1$ is proportional to $1/s^2$. The number of stages can be quite large, in which case we see that the method behaves rather like the second order explicit method.

Now we consider the general nonlinear system in autonomous form. Let $y(t)$ be a smooth solution of $y' = F(y) = F_E(y) + F_I(y)$ and let $y_n = y(t_n)$. For $\tau_n \to 0$ we find as counterpart of (11) the expansion

$$y_{n+1} = y(t_n) + \tau_n\, y'(t_n) + \frac{1}{2}\tau_n^2\, y''(t_n) + \tilde{\mu}_1 \tau_n^2\, F_I'(y(t_n))F(y(t_n)) + \mathcal{O}(\tau_n^3). \tag{12}$$

Here F_I' is the Jacobian of the reaction term. It is readily verified that the expansion for the general non-autonomous system $y' = F(t,y) = F_E(t,y) + F_I(t,y)$ extends to

$$\begin{aligned} y_{n+1} \;=\; & y(t_n) + \tau_n\, y'(t_n) + \frac{1}{2}\tau_n^2\, y''(t_n) + \tilde{\mu}_1 \tau_n^2\, F_I'(t_n, y(t_n))F(t_n, y(t_n)) + \\ & \tilde{\mu}_1 \tau_n^2 \frac{\partial F_I}{\partial t}(t_n, y(t_n)) + \mathcal{O}(\tau_n^3). \end{aligned} \tag{13}$$

We assess the size of the local truncation error with the τ_n^2 terms in this expansion,

$$Est_{n+1} = \frac{1}{2}\tau_n^2\, y''(t_n) + \tilde{\mu}_1 \tau_n^2\, F_I'(t_n, y(t_n))F(t_n, y(t_n)) + \tilde{\mu}_1 \tau_n^2 \frac{\partial F_I}{\partial t}(t_n, y(t_n))\,. \tag{14}$$

Controlling the size of the last term in a Taylor series that is taken into account by the formula goes back to the influential solvers of Zonneveld [7]. This approach to controlling error is different from that of RKC, which estimates the leading term of the local truncation error. Some discussion of the distinction is found in [4, p. 342] and references cited therein. In this approach a distinction is made between adaptation of the step size and controlling the error. A reasonable way to select the step size is to control the size of this term. The actual error made in the step is not known, but it is of higher order than the term estimated, so it is expected to be rather smaller. Indeed, if all is going well, it is considerably smaller and the control is rather conservative. This is reminiscent of local extrapolation. Because we took an approach to error control different from that found in RKC, it would be misleading to compare directly results computed by the two codes when they are both applicable.

2.3 Error Estimate

There are a number of points of interest in the implementation of (14). The terms are approximated independently by

$$\begin{aligned} y''(t_n) &\approx \frac{1}{\tau_n}\left[F(t_{n+1},y_{n+1}) - F(t_n,y_n)\right] , \\ F_I'(t_n,y(t_n))F(t_n,y(t_n)) + \frac{\partial F_I}{\partial t}(t_n,y(t_n)) &\approx \frac{1}{\tau_n}\left[F_I(t_{n+1},y_{n+1}) - F_I(t_n,y_n)\right] . \end{aligned}$$

We filter [3] the error estimate with the matrix $(I-\gamma\tau_n A)^{-1}$ so that it has the correct qualitative behavior for the stiff reaction components. The matrix A here is an approximation to the Jacobian F_I' that was computed during the evaluation of the implicit formula by a modified Newton iteration. In IRKC we take this matrix to be the Jacobian F_I' at (t_n, y_n) rather than at (t_{n+1}, y_{n+1}) because we have accepted the approximation y_n and are trying to decide whether to accept y_{n+1}.

The coefficient γ is at our disposal. It would be natural to choose it so that the matrix $I - \gamma\tau_n F_I'(t_n, y_n)$ is the same as the iteration matrix in order to reuse an LU decomposition. However, to reduce greatly the storage required by the solver, we do not save the decomposed iteration matrices. This means that we must repeat the calculations, but in partial recompense, we are then free to choose a good value for γ. Besides, we shall see that using an iteration matrix is not satisfactory in the present application.

To select a value for γ, we examine the behavior of Est_{n+1} for the scalar test equation (9),

$$Est_{n+1} = \frac{1}{2}\,\frac{z_E + (1+2\tilde{\mu}_1)z_I}{1-\gamma z_I}\,(y_{n+1} - y_n)\,.$$

As $z_I \to -\infty$ the stability function $R_s(z_E, z_I)$ remains bounded and is even of moderate size,

$$R_s(z_E, -\infty) = 1 - b_s T_s(\omega_0) + b_s T_s(0) \approx 1 - \frac{1}{3}(T_s(\omega_0) - T_s(0))\,.$$

We choose γ so that the same is true of the error estimate. Now

$$Est_{n+1} = \frac{1}{2}\,\frac{1+2\tilde{\mu}_1}{\gamma}\,b_s\left(T_s(\omega_0) - T_s(0)\right)y_n \approx \frac{1}{6}\,\frac{1+2\tilde{\mu}_1}{\gamma}\left(T_s(\omega_0) - T_s(0)\right)y_n\,.$$

If we were to reuse iteration matrices, we would take $\gamma = \tilde{\mu}_1$. With this choice $Est_{n+1} \sim s^2 y_n$ for large s because $\tilde{\mu}_1 \sim 1/s^2$. This is clearly a bad choice in a code which may resort to s-values that are quite large. We simply take $\gamma = 1$. It leads to a bounded Est_{n+1} of moderate size and suits our purpose of filtering stiff components in the error estimate.

Summarizing, the error estimator Est_{n+1} used by the new IRKC code is evaluated by solving the linear system

$$(I - \tau_n F_I'(t_n, y_n))\, Est_{n+1} = \frac{1}{2}\tau_n(F(t_{n+1}, y_{n+1}) - F(t_n, y_n)) + \tau_n \tilde{\mu}_1 (F_I(t_{n+1}, y_{n+1}) - F_I(t_n, y_n))\,. \tag{15}$$

The special form of the ODEs results in a block diagonal matrix $F_I'(t_n, y_n)$, making evaluation of this estimate efficient, especially as regards storage.

2.4 Step Size Selection

Each step size τ_n is chosen so that the user-specified tolerance is met by the estimated local error. The strategy implemented in IRKC is quite similar to the one used in RKC. The user specifies scalar relative (*rtol*) and absolute (*atol*) error tolerances. Because the solver is based on a first-order method, these tolerances should not be very small. With this in mind, we have set the default values to $rtol = 10^{-2}$ and $atol = 10^{-3}$.

The norm of the error estimate is computed in blocks. For k running over all NG = NEQN/NPDES grid points, we let $y_{n,k}$ denote the NPDES elements of y_n corresponding to the $k-$th grid point, and similarly for other quantities. For each grid point k the Euclidean norm of the weighted error vector is the square root of

$$err_k = \sum_{i=1}^{\text{NPDES}} \left(\frac{Est_{n+1,k}^{(i)}}{atol + rtol\ \max(|y_{n,k}^{(i)}|\,,\ |y_{n+1,k}^{(i)}|)} \right)^2 .$$

Once all blocks are processed, the weighted RMS norm of the error is formed as

$$\|Est_{n+1}\| = \sqrt{\frac{1}{\text{NEQN}} \sum_{k=1}^{\text{NG}} err_k}\ .$$

If $\|Est_{n+1}\| \leq 1$, the current step is accepted and the next step size is predicted using information gathered in the current and preceding step. If $\|Est_{n+1}\| > 1$, the step is rejected and retried with a smaller step size τ_n. In either case the new step size is taken to be a fraction of the largest possible value so as to reduce the number of rejected steps. Specifically, the new step size is

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

where

$$fac = 0.8 \left(\frac{\|Est_n\|^{1/2}}{\|Est_{n+1}\|^{1/2}} \frac{\tau_n}{\tau_{n-1}} \right) \frac{1}{\|Est_{n+1}\|^{1/2}}\,.$$

If the step is rejected, and also after the first successful step, the factor in parentheses is omitted. If the modified Newton iteration fails to converge, the step size is simply halved.

IRKC automatically determines an initial step size in almost the same way as RKC. The scheme begins by setting the absolute value of the initial step size, ABSH, to the maximum step size HMAX. Using a bound SPCRAD on $\rho(F_E')$, it then reduces ABSH as necessary so that SPCRAD*ABSH is no bigger than one. This step size is small enough to resolve an initial transient due to F_E, but IRKC must account for F_I as well. It is easy to compute JACNRM = $\|F_I'\|_\infty$ because we have an analytical expression for this block diagonal matrix. In IRKC we further reduce ABSH as necessary so that JACNRM*ABSH is no bigger than one. The rest of the scheme for estimating an initial step size is identical to that of RKC.

3 Software Issues

We have redesigned the user interface of RKC and exploited the capabilities of Fortran 90 to make solving ODEs easier despite solving a larger class of problems. We begin here with an outline of the suite and then provide some details. Fortran 90 makes it possible to encapsulate all information about the solution as a single structure of a derived type. This approach and the dynamic storage facilities of Fortran 90 make it possible to relieve the user of tedious details of allocating storage. The solution structure can be called anything, but let us suppose that it is called SOL. It must be declared as type IRKC_SOL. This structure and the computation itself are initialized by a call to a function IRKC_SET. Among other things, the user must specify the initial point and a final point in this call. We exploit the possibility in Fortran 90 of optional arguments so that the most common options can be set by default. The integration is done by the function IRKC. The default in IRKC_SET is for the integration to proceed a step at a time, but optionally the solver can return just the solution at the final point. After each step an auxiliary function IRKC_VAL can be used to approximate the solution anywhere in the span of the step. Statistics are available directly in fields of the solution structure, but they can all be displayed conveniently by calling the auxiliary function IRKC_STATS.

3.1 Form of the Solution

The solution structure SOL is initialized by IRKC_SET. It is then an input argument of the solver IRKC. If the integration is being done a step at a time (the default), the SOL returned at one step is input to IRKC for the next step. The solution structure has a good many fields. The most interesting fields are the current value of the independent variable, SOL%T, and the current approximate solution, SOL%Y, a vector of NEQN components. The logical quantity SOL%DONE is monitored to learn when the integration is complete.

The methods of IRKC provide a solution between mesh points that is evaluated in an auxiliary function. An approximate solution YOUT at a point TOUT in the span of the last step is obtained by YOUT = IRKC_VAL(SOL,TOUT). Some of the data held in fields of SOL for this purpose might be of direct interest, viz., the current approximation to the first derivative of the solution, SOL%YP, and the size of the last step taken, SOL%HLAST.

A convenient way to see all the statistics is to CALL IRKC_STATS(SOL). Individual statistics are available as the integer fields

SOL%NFE	number of evaluations of F_E
SOL%NFI	number of evaluations of F_I
SOL%NSTEPS	number of steps
SOL%NACCPT	number of accepted steps
SOL%NREJCT	number of rejected steps
SOL%NFESIG	number of function evaluations used in estimating $\rho(F'_E)$
SOL%MAXM	maximum number of stages used

3.2 Specification of the Task

Only a few quantities are required to specify the task and initialize the integration. This is done with a call

```
SOL = IRKC_SET(T0,Y0,TEND)
```

This says that the integration is to start at T0 with a vector Y0 of NEQN components as initial value and go to TEND. It also says that the solution structure is to be called SOL. These arguments must appear, and in the order shown, but the remaining, optional arguments can follow in any order because they are specified using keywords.

The solver must be told how many PDEs there are. This is 1 by default and any other value must be supplied with the keyword NPDES. For example, if there are 3 PDEs, the call above is changed to

```
SOL = IRKC_SET(T0,Y0,TEND,NPDES=3)
```

The most commonly used options are the error tolerances. RKC provides for a scalar relative error tolerance and a vector of absolute error tolerances. To leading order the storage required is a multiple of NEQN. A guiding principle of RKC, and especially IRKC, is minimize this multiple. Quite often users have scaled the variables so that a scalar absolute error tolerance is appropriate. If not, they can always rescale the variables so as to do away with the need for an array of NEQN absolute error tolerances. Accordingly, we have chosen to implement only a scalar absolute error tolerance in IRKC. This decision makes the solver easier to use and simplifies the program. The default relative error tolerance is 10^{-2} and the default absolute error tolerance is 10^{-3}. The call just illustrated causes the solver to use these default values. Other values are imposed with the keywords RE and AE. For example, to use a relative error tolerance of 10^{-3} and the default absolute error tolerance, the call is changed to

```
SOL = IRKC_SET(T0,Y0,TEND,NPDES=3,RE=1D-3)
```

By default IRKC takes one step at a time, but it can be instructed to return only after reaching TEND by giving the keyword ONE_STEP the value .FALSE. The methods for handling the explicit term F_E involve an approximate bound on the spectral radius of the Jacobian F_E'. This can be supplied with a function in a manner described below, but the default is to have the solver compute a bound. The cost of doing this can be reduced substantially when the Jacobian is constant by giving the keyword CONSTANT_J the value .TRUE. Sometimes it is useful to impose a maximum step size. This is done with the keyword HMAX. As described in Section 2.4, IRKC selects an initial step size automatically, but a value can be supplied with the keyword H0.

It is sometimes useful to reset quantities and continue integrating. This is done by replacing the initial data T0,Y0 in the call list of IRKC_SET with the solution structure SOL. If the required argument TEND or any of the optional arguments are changed, the new values replace those stored in SOL and SOL%DONE is changed to .FALSE. Of course, NPDES cannot be changed. For example, we can instruct the solver to quit returning at every step with

```
SOL = IRKC_SET(SOL,TEND,ONE_STEP=.FALSE.)
```

and then call the solver again to continue on to TEND.

Unless instructed to the contrary, the solver will print a message and stop when a fatal error occurs. This response to a fatal error can be changed with the optional argument STOP_ON_ERROR. Setting it to .FALSE. will cause the solver to return with a positive value of the integer SOL%ERR_FLAG that indicates the nature of the error. In this way a user can prevent an unwelcome termination of the run. Of course, if this report of a fatal error is ignored and the solver is called again with a positive value of SOL%ERR_FLAG, it will print a message and stop.

3.3 Defining the Equations

The differential equations are supplied as subroutines to IRKC. A typical invocation of IRKC looks like

```
CALL IRKC(SOL,F_E,F_I)
```

All three arguments in this call are required. The solution structure SOL is used for both input and output. F_E is the name of a subroutine for evaluating the explicit part of (2). It is supplied just as for RKC, i.e., there is a subroutine of the form F_E(NEQN,T,Y,DY) that accepts a vector Y of length NEQN, evaluates $F_E(t, y)$, and returns it as a vector DY of length NEQN. As explained previously, the term to be handled implicitly must be defined in a particular way that we now discuss more fully.

Let NPDES be the number of PDEs. The ODEs must be coded in blocks that correspond to grid points. Specifically, for I = 1, ..., the equations with indices I:I+NPDES-1 must correspond to a single grid point. Accordingly there is to be a subroutine of the form F_I(GRID_POINT,NPDES,T,YG,DYG,WANT_JAC,JAC). For an index I = GRID_POINT, it evaluates F_I(T,YG) for YG = Y(I:I+NPDES-1). This value is returned as a vector DYG of NPDES components. The solver also requires the Jacobian of this function, a matrix JAC of NPDES by NPDES entries. It is convenient to have this computation in the subroutine where F_I is evaluated, but it does not have to be done every time that F_I is evaluated. The logical variable WANT_JAC is used to inform the subroutine when JAC is to be formed along with DYG.

By default the solver approximates the spectral radius $\rho(F_E')$ using a nonlinear power method. This is convenient and often works well, but it can fail. When it is not too much trouble to supply a function that returns an upper bound for $\rho(F_E')$ of roughly the correct size, this should be done because the integration is then faster, more reliable, and uses less storage. This function must have the form SR(NEQN,T,Y). Its name is supplied to the solver as an optional fourth argument, but in the circumstances there is no point to using a keyword, so the call has the form

```
CALL IRKC(SOL,F_E,F_I,SR)
```

4 Numerical Examples

In this section the use of IRKC is illustrated by means of two test problems. The Fortran 90 code for IRKC as well as for the accompanying examples can be obtained by anonymous ftp from the address ftp://ftp.cwi.nl/pub/bsom/irkc. IRKC can also be downloaded from netlib@ornl.gov (send irkc.f90 from ode). To keep the codes for the test examples as transparent as possible, we choose problems in one spatial dimension. This by no means implies that IRKC encounters difficulties in solving problems in more spatial dimensions. To the contrary, the advantages of IRKC compared with an implicit solver are more pronounced in more dimensions. This is because the stiffness in the explicit part increases slowly with the number of spatial dimensions whereas the stiffness of the part handled implicitly is confined to individual grid points and so is independent of the number of dimensions.

Example 1. We begin with the solution of a system of ODEs that arises from semidiscretization of the reaction-diffusion equation

$$u_t = u_{xx} + (1-u)u^2 \quad \text{for} \quad 0 \le t \le 10, \quad 0 \le x \le 10 \tag{16}$$

with initial and Dirichlet boundary values

$$u(x,0) = 10(10-x), \;\; u(0,t) = 100, \;\; u(10,t) = 0.$$

The term u_{xx} is approximated by second-order central differences on a mesh $\{x_m\}$ with equal spacing of h to get $F_E(t,y)$ and the other term is evaluated at mesh points to get $F_I(t,y)$. If $y_m(t) \approx u(x_m,t)$, a typical component of F_E is $\left(y_{i-1}(t) - 2y_i(t) + y_{i+1}(t)\right)/h^2$ and a typical component of F_I is $\left(1-y_i(t)\right) y_i^2(t)$. Evidently F'_E is a constant, tridiagonal matrix and F'_I is a diagonal matrix with typical entry $\left(2-3y_i(t)\right) y_i(t)$. Using either Gershgorin's theorem or $\|F'_E\|_\infty$, we obtain easily the bound $4/h^2$ for $\rho(F'_E)$. For this simple example there is an analytical expression for the eigenvalues which shows that the upper bound is very close to $\rho(F'_E)$. In the example program there are 50 ODEs ($h = 10/51$) and the bound is about 104. On a time interval of length 10, we see that the diffusion part causes only moderate stiffness. When the solver itself estimates a bound on the spectral radius, it computes values that range from 123 to 124. The solver computes a bound by estimating the spectral radius and, for safety reasons, increasing the estimate by 20%. For this example the solver has evidently computed a good estimate of $\rho(F'_E)$. Because of the Dirichlet boundary condition $u(0,t) = 100$, there is always an entry in F'_I, hence an eigenvalue of this diagonal matrix, that is approximately $-3\ 10^4$. We see that on a time interval of length 10, the reaction part of the PDE causes the ODEs to be stiff. Accordingly, the example program computes solution profiles at times $0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10$ to show the initial transient as well as illustrate how to obtain output at specific points. The (exact ODE) solutions at these output points are shown in Figure 1. Here, the facility offered by IRKC_VAL(SOL,TOUT) has been used to obtain solution values at the intermediate output points. The solution structure SOL was initialized as

```
SOL = IRKC_SET(0D0,Y,10D0,RE=TOL,AE=TOL)
```

i.e., we took both the absolute and relative error tolerance equal to a parameter *tol*. For several values of *tol*, the integration results are collected in Table 1. These results are obtained by

```
CALL IRKC(SOL,F_E,F_I,SR)
```

where the function SR provides the solver with the estimate $4/h^2$ for the spectral radius of F'_E. As we can see from this table, this rather simple problem has been integrated by IRKC without difficulties. Because the subroutine F_I is called for each grid point, the total number of calls reported in Table 1 has been scaled by the number of grid points (NEQN/NPDES). The ratio of the number of calls to F_I and F_E is an indication of the average number of modified Newton iterations. This number is seen to be slightly larger than 2.5. Because the stopping criterion in the modified Newton process is related to the tolerance for the error control, the number of iterations does not necessarily decrease when the tolerance is reduced. Notice that the global error in $t = 10$, measured in the L_2-norm, refers to the time integration error only. For an accurate solution of the PDE (16), the spatial resolution needs refinement as well.

Example 2. Our second example is a more substantial problem. It originates in radiation-diffusion theory and is a 1D version of a 2D problem described in [2]. Here we remark that the original 2D problem has been discussed in our algorithm paper [6], where we have used a preliminary version of the IMEX code to solve this problem (see also [1] where the 2D problem has been solved with RKC). Again, the reduction to 1D is motivated only by illustrative arguments; the characteristic difficulties in this problem are maintained in the 1D version.

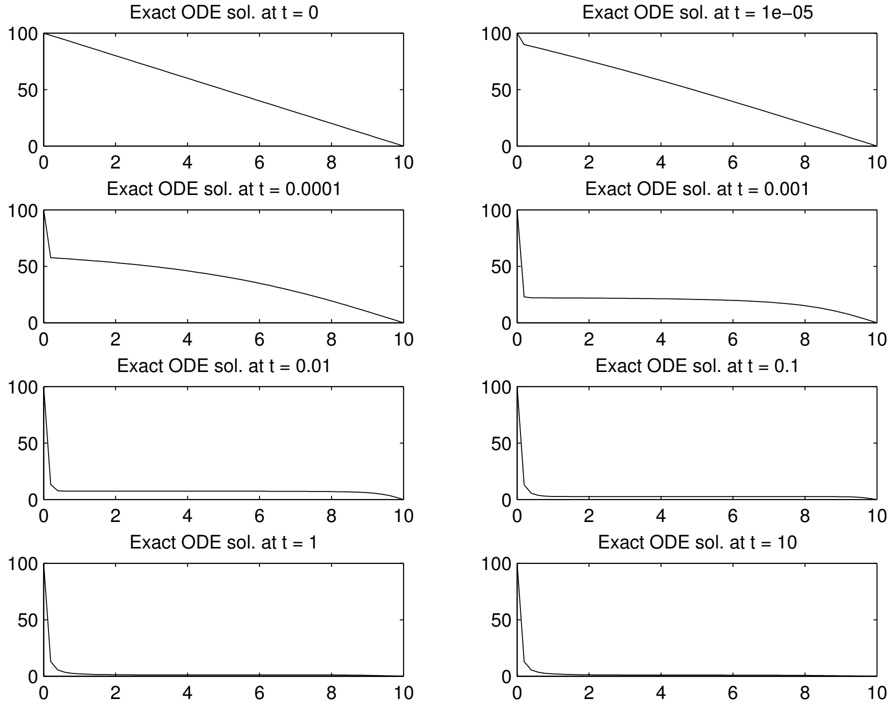

Figure 1: Example 1: Reference solution at several points in time.

Table 1: Example 1: Results obtained at $t = 10$ with grid spacing h=10/51.

Quantity	$tol = 10^{-2}$	$tol = 10^{-3}$	$tol = 10^{-4}$
Number of successful steps	99	328	1062
Number of rejected steps	2	2	2
Evaluations of F_E	413	1139	3374
Evaluations of F_I / (NEQN/NPDES)	1035	2970	8936
Maximum number of stages used	20	16	11
Global error (discrete L_2-norm)	1.03 10^{-3}	1.49 10^{-4}	4.07 10^{-5}

This test problem consists of two strongly nonlinear diffusion equations with a highly stiff reaction term (an idealization of non-equilibrium radiation diffusion in a material). The dependent variables E and T represent radiation energy and material temperature, respectively. One source of such problems is laser fusion. The equations are defined on the unit interval in space for $t > 0$,

$$\begin{aligned} E_t &= (D_1\, E_x)_x \;+\; \sigma(T^4 - E)\,, \\ T_t &= (D_2\, T_x)_x \;-\; \sigma(T^4 - E)\,, \end{aligned} \tag{17}$$

with

$$\sigma \;=\; \frac{Z^3}{T^3}\,, \qquad D_1 \;=\; \frac{1}{3\sigma + |E_x|/E}\,, \qquad D_2 \;=\; k\,T^{5/2}\,.$$

Here $Z = Z(x)$ represents the atomic mass number which may vary in the spatial interval to

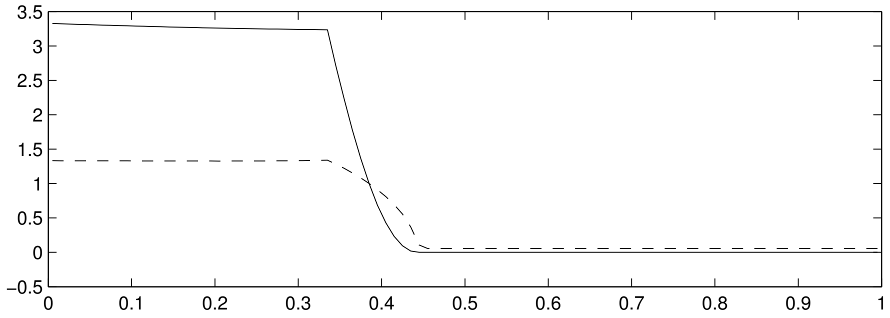

Figure 2: Example 2: Reference solution at $t = 3$ with grid spacing h=1/100: radiation energy E (solid line) and temperature T (dashed line).

reflect inhomogeneities in the material. We suppose that the temperature diffusion coefficient $k = 0.005$ and $Z(x)$ varies as

$$Z(x) = \begin{cases} Z_0 & \text{if } |x - \frac{1}{2}| \le \frac{1}{6} \\ 1 & \text{otherwise}\,. \end{cases}$$

Here we take $Z_0 = 10$. The initial values are constant in space,

$$E(x,0) \;=\; 10^{-5}\,, \qquad T(x,0) \;=\; E(x,0)^{1/4} \;\approx\; 5.62\;10^{-2}\,,$$

and the boundary conditions are

$$\begin{aligned} \tfrac{1}{4}E \,-\, \tfrac{1}{6\sigma}E_x \;&=\; 1 \qquad && \text{at } x = 0\,, \\ \tfrac{1}{4}E \,+\, \tfrac{1}{6\sigma}E_x \;&=\; 0 \qquad && \text{at } x = 1\,, \\ T_x \;&=\; 0 \qquad && \text{at } x = 0, 1\,. \end{aligned}$$

The solution of this problem has a steep temperature front moving to the right. For $Z_0 > 1$, as it is here, the movement is hampered at the interior of the interval. The radiation energy E is almost equal to T^4 except near the front where it is slightly smaller.

We discretized the PDEs in space using a uniform cell-centered grid with grid size $h = 1/100$ by means of a second-order central conservative scheme; details can be found in [1, Chapter V]. In principle this is straightforward, but the program is somewhat complicated because of special cases and having to approximate the derivative in the diffusion coefficient D_1. The resulting system of ODEs is of dimension $2/h$, so NEQN = 200. Integrating these ODEs accurately in time to $t = 3$ gives the reference solution displayed in Figure 2. Though we have chosen a grid spacing that gives an adequate resolution of the solution of the PDEs, it should be understood that it is the solution of the system of 200 ODEs that was computed accurately for the purpose of illustrating the code, not the solution of the PDEs.

The nonlinear power method used by default to calculate a bound on the spectral radius $\rho(F'_E)$ fails to converge for this strongly nonlinear problem. However, following heuristic arguments, it is possible to derive an upper bound for $\rho(F'_E)$. To that end we apply Fourier–von Neumann analysis with the diffusion coefficients D_1 and D_2 frozen at their maximal value. Here we use some information about the solution. Observing that the temperature T is approximately in the range $[0.056, 1.35]$ (see Figure 2), we arrive at the following bounds for the diffusion coefficients

$$D_1 \le \frac{1}{3\sigma} = \frac{T^3}{3Z^3} < \frac{1}{Z^3} \le 1\,, \qquad D_2 = kT^{5/2} \ll 1\,.$$

Choosing $D_1 = D_2 = 1$ in the Fourier–von Neumann analysis, we conclude that $\rho(F'_E)$ is bounded by $4/h^2 = 40,000$.

It is illuminating to estimate the stiffness caused by the reaction terms. It is easily verified that

$$F'_I(E,T) = \begin{pmatrix} -\alpha & \beta \\ \alpha & -\beta \end{pmatrix}, \quad \text{with} \quad \alpha = \frac{Z^3}{T^3}\,, \quad \beta = Z^3(1 + \frac{3E}{T^4})\,.$$

The eigenvalues of F'_I are 0 and $-(\alpha+\beta)$. Since $E \approx T^4$, $\alpha+\beta$ is dominated by the term Z^3/T^3. Again using information about the solution, we see that $\rho(F'_I)$ is roughly 6000 $Z_0^3 = 6\ 10^6$. This is quite large compared to the length of the interval of integration, so the problem is stiff and an explicit treatment of the reaction terms is impractical.

We integrated the system of the two semi-discretized PDEs from time 0 to 3 and asked for output only at $t = 3$. Again, we took both relative and absolute error tolerances to be equal to a parameter *tol*. The solver is told of all this with

```
SOL = IRKC_SET(0D0,Y,3D0,RE=TOL,AE=TOL,NPDES=2,ONE_STEP=.FALSE.)
```

We coded the evaluation of the explicit and implicit terms as subroutines called F_E and F_I, respectively, and the upper bound on the spectral radius as a function called SR. The solver is then invoked with

```
CALL IRKC(SOL,F_E,F_I,SR)
```

The results of the integration are presented in Table 2 for several values of *tol*. For this example, we see that the ratio of the number of calls to F_I and F_E is slightly larger than 2.

Table 2: Results obtained at $t = 3$ for test problem (17).

Quantity	*tol* $= 10^{-2}$	*tol* $= 10^{-3}$	*tol* $= 10^{-4}$
Number of successful steps	72	220	675
Number of rejected steps	20	34	3
Evaluations of F_E	4133	7020	10840
Evaluations of F_I / (NEQN/NPDES)	8369	14576	24305
Maximum number of stages used	135	82	44
Global error in E (discrete L_2-norm)	$7.24\ 10^{-4}$	$2.03\ 10^{-4}$	$1.75\ 10^{-4}$
Global error in T (discrete L_2-norm)	$2.15\ 10^{-3}$	$2.14\ 10^{-4}$	$1.79\ 10^{-4}$

Acknowledgements B.P. Sommeijer and J.G. Verwer acknowledge support from the Dutch BSIK/BRICKS project.

References

[1] W. Hundsdorfer and J.G. Verwer (2003), *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, Springer Series in Computational Mathematics, Vol. 33, Springer, Berlin.

[2] V.A. Mousseau, D.A. Knoll, and W.J. Rider (2000), *Physics-based preconditioning and the Newton–Krylov method for non-equilibrium radiation diffusion*, J. Comput. Phys. 160, pp. 743–765.

[3] L.F. Shampine and L.S. Baca (1984), *Error estimators for stiff differential equations*, J. Comput. Appl. Math. 11, pp. 197–207.

[4] L.F. Shampine (1994), *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York.

[5] B.P. Sommeijer, L.F. Shampine, and J.G. Verwer (1997), *RKC: An explicit solver for parabolic PDEs.* J. Comput. Appl. Math. 88, pp. 315–326.

[6] J.G. Verwer and B.P. Sommeijer (2003), *An implicit-explicit Runge–Kutta–Chebyshev scheme for diffusion-reaction equations.* SIAM J. Sci. Comput. 25, pp. 1824–1835.

[7] J.A. Zonneveld (1964), *Automatic Numerical Integration*, Mathematisch Centrum, Amsterdam.