
ELSEVIER

Journal of Computational and Applied Mathematics 60 (1995) 309–329

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Parallel iteration across the steps of high-order Runge–Kutta methods for nonstiff initial value problems☆

P.J. van der Houwen*, B.P. Sommeijer, W.A. van der Veen

Afd. Numerieke Wiskunde, CWI, P.O. Box 94079, 1090 GB Amsterdam, Netherlands

Received 23 August 1993; revised 6 April 1994

Abstract

For the parallel integration of nonstiff initial value problems (IVPs), three main approaches can be distinguished: approaches based on "parallelism across the problem", on "parallelism across the method" and on "parallelism across the steps". The first type of parallelism does not require special integration methods and can be exploited within any available IVP solver. The method-parallelism approach received much attention, particularly within the class of explicit Runge–Kutta methods originating from fixed point iteration of implicit Runge–Kutta methods of Gaussian type. The construction and implementation on a parallel machine of such methods is extremely simple. Since the computational work per processor is modest with respect to the number of data to be exchanged between the various processors, this type of parallelism is most suitable for shared memory systems. The required number of processors is roughly half the order of the generating Runge–Kutta method and the speed-up with respect to a good sequential IVP solver is about a factor 2. The third type of parallelism (step-parallelism) can be achieved in any IVP solver based on predictor–corrector iteration and requires the processors to communicate after each full iteration. If the iterations have sufficient computational volume, then the step-parallel approach may be suitable for implementation on distributed memory systems. Most step-parallel methods proposed so far employ a large number of processors, but lack the property of robustness, due to a poor convergence behaviour in the iteration process. Hence, the effective speed-up is rather poor. The dynamic step-parallel iteration process proposed in the present paper is less massively parallel, but turns out to be sufficiently robust to achieve speed-up factors up to 15.

Keywords: Numerical analysis; Runge–Kutta methods; Parallelism

1. Introduction

The last five years have shown an increased interest in solving the initial value problem (IVP)

$$y'(t) = f(y(t)), \quad y(t_0) = y_0, \quad y, f \in \mathbb{R}^d \tag{1.1}$$

☆ The research reported in this paper was partly supported by STW (Netherlands Foundation for the Technical Sciences).

* Corresponding author.

SSDI 0377-0427(94)00047-5

on parallel computers. One of the classes of parallel IVP solvers for nonstiff problems that received relatively much attention is the class of predictor–corrector (PC) methods based on Runge–Kutta (RK) correctors (see, e.g., [2–4, 19, 20, 10–14, 17]). As was observed in [11], PC iteration (and in fact, all functional iteration methods), when applied to RK correctors, possess automatically parallelism across the components of the stage vector iterates, because these components can be iterated in parallel. Therefore, we shall henceforth refer to these methods as *PIRK methods* (parallel iterated Runge–Kutta methods [19]).

Highly accurate correctors are provided by the classical, collocation-based RK methods such as the Gauss methods (sometimes called the Kuntzmann–Butcher methods [9] or the Butcher–Kuntzmann methods [20], and in this paper referred to as BK methods). Moreover, automatic stepsize variation and predictor formulas can be easily obtained by means of the collocation polynomial. In [19] numerical results obtained by the PIRK method using the 5-point BK corrector were reported. This PIRK method, equipped with the last-step value (LSV) predictor and a simple stepsize strategy, already halves the sequential costs when compared with the highly efficient, sequential DOPRI8 code [9].

However, the number of iterations needed to achieve the corrector accuracy is still high (about the order of the corrector). In order to reduce the number of iterations, we introduced in [20] preconditioning in the PIRK method and found that the number of iterations reduces substantially (cf. [20]). For example, for the often used Arenstorf test problem (cf. [9, p. 127]), preconditioned PIRK based on the PC pair consisting of the extrapolation (EXP) predictor and the 4-point BK corrector showed an averaged speed-up factor of 4.4 with respect to DOPRI8 in the accuracy range of 3 to 8 correct digits. An interesting feature of the iterated RK methods is the highly efficient performance of the high-order correctors, also in the low accuracy range. As an illustration, we applied the preconditioned {EXP, 13-point BK} PC pair to the Arenstorf problem, and found an averaged speed-up factor of 6.7 with respect to DOPRI8, again in the accuracy range of 3 to 8 correct digits.

In this paper, we try to reduce the sequential costs by applying "parallelism across the steps" to the PIRK methods. In some sense, our approach shows similarities with that of Miranker and Liniger [15] and of Nievergelt [16], but is most closely related to the approach of the Trieste group (see [1]). The main difference with the Trieste approach is a more robust iteration process (Gauss–Seidel type instead of Steffenson), however, at the cost of less massive parallelism. Nevertheless, our numerical experiments show that the particular type of PIRK methods Across the Steps (PIRKAS methods) developed in this paper often require not more than two sequential function calls per step for solving the corrector and give rise to speed-up factors up to 15 when compared with the best sequential codes available (i.e., DOPRI8). We shall confine our considerations to PIRKAS methods *without* preconditioning. Introducing preconditioning and extension to stiff initial value problems will be subject of future research.

2. Parallelism across the steps

We consider implicit, s-stage RK methods written in the form of an $(s+1)$-stage General Linear Method (GLM), introduced in [5] (see also [6, p. 340]):

$$Y_n = (E \otimes I)Y_{n-1} + h(B \otimes I_d)F(Y_n), \quad n = 1, \ldots, N. \tag{2.1a}$$

Here h denotes the stepsize, the matrix B contains the RK parameters, and $F(Y_n)$ contains the derivative values $(f(Y_{n,i}))$, where $Y_{n,i}$ denote the d-dimensional components of the (extended) stage vector Y_n (because of the GLM representation (2.1a), the RK solution at the step points is lumped into Y_n). It will be assumed that (2.1a) possesses s implicit stages and one explicit stage. The component of Y_n corresponding to the explicit stage approximates the exact solution at the step point $t_n = t_{n-1} + h$. The other stage vector components $Y_{n,i}$ represent numerical approximations at the intermediate points $t_{n-1} + c_i h$, where $c = (c_i) = Be$, e being the vector with unit entries. In the sequel we assume that the last stage is the explicit one, so that the matrices E and B take the form

$$E := \begin{pmatrix} 0 & \dots & 0 & 1 \\ . & \dots & . & . \\ . & \dots & . & . \\ . & \dots & . & . \\ 0 & \dots & 0 & 1 \end{pmatrix}, \quad B := \begin{pmatrix} A & \mathbf{0} \\ b^{\mathrm{T}} & 0 \end{pmatrix}, \tag{2.1b}$$

where A and b present the familiar arrays appearing in the Butcher tableau representation of RK methods. Furthermore, the matrix I is the d-by-d identity matrix, $\otimes$ denotes the Kronecker product, and we define $Y_0 = e \otimes y_0$. In the following, the dimension of I and e may change, but will always be clear from the context.

Eq. (2.1), henceforth referred to as the corrector, can be solved by the conventional PC iteration method which in a programming-like language reads

FOR $n := 1$ **TO** N
  FOR $j := 1$ **TO** m
    $Y_n^{(j)} = (E \otimes I) Y_{n-1}^{(m)} + h(B \otimes I) F(Y_n^{(j-1)})$, (2.2)

where m is the number of iterations, $Y_0^{(m)} = e \otimes y_0$, and $Y_n^{(0)}$ is to be provided by a predictor formula. Evidently, if (2.2) converges, then it converges to the corrector solution Y_n.

As mentioned in Section 1, the PC method (2.2) has been extensively analysed in a number of papers and was called a parallel iterated RK method (PIRK method) in [19] (see also [9, p. 259]). It possesses parallelism *within the iterations* (that is, for each n and j, the components of $Y_n^{(j)}$ can be evaluated in parallel), but, apart from parallelism across the problem, it does not have any further parallelism. Hence, the total computational effort consists of Nm evaluations of a full derivative vector $F(Y_n^{(j-1)})$, but on a computer possessing s processors, the sequential costs of one full derivative vector evaluation consists of evaluating just one right-hand side function f of dimension d. We shall measure the sequential costs of an *explicit* method by the total number of sequential *right-hand side evaluations*, where we tacitly assume that sufficiently many processors are available. Thus, the sequential computational complexity of the PC method (2.2) is given by $N_{\mathrm{seq}} = Nm$.

In order to increase the degree of parallelism in PIRK methods, we have to modify the recursion (2.2). The most obvious approach to achieve a high degree of parallelism in IVP methods writes the corrector (2.1) in the form $G(Y) = \mathbf{0}$, where Y represents the vector containing *all numerical approximations in the whole integration interval*, and solves this system for Y by some iteration type process. This type of parallelism has been considered by several authors (e.g., see [16, 1]). In the case of the RK solver (2.1), Y represents the N stage vectors Y_n, $n = 1, \dots, N$.

The most simple iteration process for solving $G(Y) = 0$ can be obtained from (2.2) by interchanging the loops for n and j in (2.2):

$$\begin{aligned} &\textbf{FOR } j := 1 \textbf{ TO } m \\ &\quad \textbf{FOR } n := 1 \textbf{ TO } N \\ &\qquad Y_n^{(j)} = (E \otimes I) Y_{n-1}^{(j-1)} + h(B \otimes I) F(Y_n^{(j-1)}). \end{aligned} \tag{2.3}$$

Here, we have $Y_0^{(j)} = e \otimes y_0$ for $j = 0, \ldots, m-1$. In view of load balancing of the processors, we want the sequential computational effort involved with the computation of a single iterate $Y_n^{(j)}$ to be equal for all iterates. Therefore, here and in the following, the costs of computing the prediction $Y_n^{(0)}$ are assumed to be negligible. Thus, given the initial guesses $Y_n^{(0)}$, $n = 1, \ldots, N$, first all stage vectors $Y_n^{(1)}$ are computed concurrently, then all $Y_n^{(2)}$, and so on. Hence, having sN processors available, the sequential computational complexity of the method (2.3) is given by $N_{\text{seq}} = m$. Method (2.3) resembles Jacobi-type iteration and may be considered as a PIRK method employing iteration Across the Steps of Jacobi-type (PIRKAS J method). A drawback of this seemingly "cheap" method is its slow convergence or even divergence, due to a poor first iterate $Y_n^{(1)}$, a situation that can easily occur in the case of large integration intervals. This is caused by the fact that the prediction $Y_n^{(0)}$ is either based on mere extrapolation of the initial value y_0 or just an initial guess to be provided by the user (note that predictions based on derivative information on preceding step points would increase the sequential costs by an amount of $O(N)$). As a consequence, Jacobi-type iteration is only feasible when applied on subintervals (windows). Of course, for w windows, the sequential costs will increase to $N_{\text{seq}} = wm$.

An alternative to Jacobi-type iteration is a more powerful iteration process. When applied using the window-strategy just mentioned, we may hope to reduce the number of iterations m to such an extent that the sequential costs $N_{\text{seq}} = wm$ are acceptable. In the literature, Steffenson iteration and Newton-type iteration have been considered. Full details of the Steffenson process applied to a general class of IVP solvers may be found in the papers of Bellen and his coworkers [1]. For a discussion of Newton-type iteration, we refer to the thesis of Chartier [7].

In the present paper, we shall study Gauss–Seidel type iteration processes for solving the corrector Eq. (2.1). Gauss–Seidel iteration possesses a lower degree of intrinsic parallelism than Jacobi and Steffenson iteration, but it allows us to compute a much more accurate first iterate $Y_n^{(1)}$.

2.1. The PIRKAS GS method

Consider the recursion

$$Y_n^{(j)} = (E \otimes I) Y_{n-1}^{(j)} + h(B \otimes I) F(Y_n^{(j-1)}), \quad j = 1, 2, \ldots, m; \; n = 1, 2, \ldots, N. \tag{2.4}$$

The only difference with the recursion in the PIRKAS J method (2.3) is the superscript in the first term of the right-hand side. By this modification we introduce a dependency in the time direction and therefore (2.4) may be considered as a Gauss–Seidel-type iteration process for solving (2.1). The iterates defined by (2.4) can be computed according to various orderings. Representing the iterates $Y_n^{(j)}$ by points in the (n, j)-plane, we may compute them row-wise (j constant) or column-wise (n constant) or diagonal-wise ($n + j$ constant). We emphasize that the solutions resulting from these orderings are algebraically equivalent. However, from an implementational point of view, of all

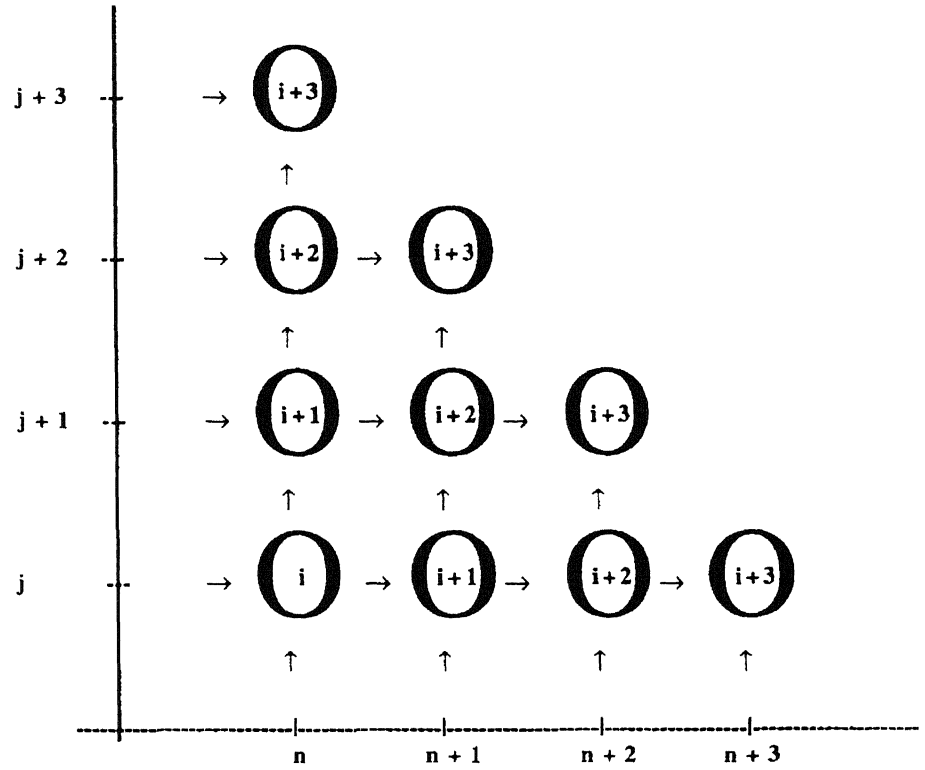

Fig. 1. Grid of $Y_n^{(j)}$ iterates in the (n, j)-plane.

orderings of computation allowed by (2.4), the diagonal-wise ordering possesses maximal parallelism, because all iterates $Y_n^{(j)}$ with $n + j$ constant can be computed concurrently (see Fig. 1). Thus, in the diagonal-wise ordering we first compute the iterates labeled by i, next the iterates labeled $i + 1$, etc. Notice that an iterate can be computed as soon as its left and lower neighbours are available. In comparison with Jacobi iteration, the intrinsic parallelism is reduced considerably, but this is compensated by a much faster convergence.

In the following, we shall analyse and evaluate the performance of the Gauss–Seidel type PIRKAS method (2.4) (briefly PIRKAS GS method). In accuracy and stability considerations, it will sometimes be convenient to assume that the iterates are computed by the row-wise ordering. However, in actual computation, we of course employ the diagonal-wise ordering.

Fig. 1 suggests introducing the step index $i = n + j$, where n and j are the time index and iteration index, respectively, and writing the correction formula (2.4) as

$$Y_n^{(i-n)} = (E \otimes I)Y_{n-1}^{(i-n)} + h(B \otimes I)F(Y_n^{(i-n-1)}). \tag{2.5}$$

The corresponding computational scheme can be implemented according to

FOR $i := 1$ **TO** $m + 1$
  FOR $n := 0$ **TO** i
    CALL correction (i, n)

FOR $i := m + 2$ **TO** $N + m$
  FOR $n := i - m$ **TO** $\min\{i, N\}$
    CALL correction (i, n) (2.6a)

where the subroutine correction (i, n) is defined by

IF $i = n$ **THEN** Compute $Y_n^{(i-n)} = Y_n^{(0)}$ by means of the predictor formula (see Section 2.2)
ELSE **IF** $n = 0$ **THEN** $Y_n^{(i-n)} = e \otimes y_0$ (2.6b)
ELSE Compute $Y_n^{(i-n)}$ by means of the correction formula

The method {(2.5), (2.6)} will be referred to as the PIRKAS GS method. The sequential costs are $N_{seq} = N + m$ right-hand side evaluations on sm processors (see Table 1). We remark that the parallelism within the correction formula is fine grain compared with the parallelism across the steps. Hence, the correction formula (2.6b) is most suitably implemented on a shared memory unit, whereas the scheme (2.6a) seems to be more efficient for implementation on distributed memory systems. For example, the appropriate architecture would be a network of m workstations, each having s shared memory processors.

2.2. Regions of stability and convergence

In discussing convergence and stability, it is convenient to assume that the iterates are computed according to the row-wise ordering (cf. the discussion at the beginning of the previous section). Thus, we assume that first all iterates $Y_n^{(1)}$, $n = 1, \ldots, N$ are computed, next $Y_n^{(2)}$, $n = 1, \ldots, N$, etc. In order to get insight into the (linear) stability region and the convergence region of the PIRKAS GS method, we consider the test equation $y'(t) = \lambda y(t)$, where λ is assumed to run through the spectrum of $\partial f/\partial y$. With respect to this test equation, the linear stability properties of the PIRKAS GS method are determined by the convergence properties of the iteration process, the stability of the corrector and the stability of the first iterates $Y_n^{(1)}$, $n = 1, 2, \ldots, N$. Assuming that the underlying corrector is A-stable, the stability region of the PIRKAS GS method is the intersection of the stability region of the formula defining the first iterate $Y_n^{(1)}$ and the region of convergence of the correction formula (2.4).

Table 1
Computational scheme for the PIRKAS GS method (2.6)

i	t_1	t_2	t_3	...	t_m	t_{m+1}	...	t_{N-m}	t_{N-m+1}	...	t_N
2	$Y_1^{(1)}$										
3	$Y_1^{(2)}$	$Y_2^{(1)}$									
⋮	⋮	⋮	⋮	⋮	⋮						
$m+1$	$Y_1^{(m)}$	$Y_2^{(m-1)}$	$Y_3^{(m-2)}$	⋮	$Y_m^{(1)}$						
$m+2$		$Y_2^{(m)}$	$Y_3^{(m-1)}$	⋮	$Y_m^{(2)}$	$Y_{m+1}^{(1)}$					
⋮			⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
$N+1$									$Y_{N-m+1}^{(m)}$	⋮	$Y_N^{(1)}$
⋮										⋮	⋮
$N+m$											$Y_N^{(m)}$

2.2.1. Stability region of the first iterate

We restrict our considerations to *one-step* predictors based on information from the preceding interval $(t_{n-2}, t_{n-1}]$, that is, $Y_n^{(1)}$ is computed by means of information coming from the iterate $Y_{n-1}^{(1)}$. As already observed, we want all iterations of comparable sequential computational complexity, so that we are led to the predictor formula $Y_n^{(0)} = (E_n^* \otimes I)Y_{n-1}^{(1)}$, to obtain

$$Y_n^{(1)} = (E \otimes I)Y_{n-1}^{(1)} + h(B \otimes I)F((E_n^* \otimes I)Y_{n-1}^{(1)}), \quad n = 1,2,\ldots,N, \tag{2.7}$$

where E_n^* is a still free, $(s+1)$-by-$(s+1)$ extrapolation matrix. Obviously, this formula should be a sufficiently stable step-by-step method by itself. Thus, the situation is different from that in conventional PC methods where only accuracy plays a role, because in that case the corrector is (numerically) solved before advancing to the next step point.

The most simple choice for the free matrix E_n^* in (2.7) sets $E_n^* = E$ for all n (LSV predictor). The resulting method (2.7) reduces to the explicit Euler method for the successive components of $Y_n^{(1)}$, the stability region of which is well known.

An alternative to the "trivial" choice $E_n^* = E$ is to exploit the fact that the underlying corrector is based on the collocation principle. This means that the components $Y_{n,i}$ are approximations to the exact solution at $t_{n-1} + c_i h$ of (at least) order s. Hence, extrapolating the collocation polynomial through the values $Y_{n-1,i}^{(1)}$ yields predictions $Y_{n,i}^{(0)}$ of the same (local) order. The corresponding predictor will be referred to as the EXP predictor. The order conditions for the EXP predictor are given by

$$E_n^*(c-e)^k = (r_n c)^k, \quad r_n := \frac{h_n}{h_{n-1}}, \quad h_n := t_n - t_{n-1}, \quad k = 0,1,\ldots,s,$$

which uniquely define the matrix E^*. It can explicitly be expressed in the form

$$E_n^* = VU^{-1}, \qquad U := (e, (c-e), \ldots, (c-e)^s), \qquad V := (e, r_n c, \ldots, (r_n c)^s).$$

The stability region of (2.7) is obtained by applying it to the test equation with constant stepsize h, to obtain

$$Y_n^{(1)} = EY_{n-1}^{(1)} + zBE^*Y_{n-1}^{(1)}, \quad z := \lambda h.$$

Hence, the stability region of (2.7) consists of the points z where the eigenvalues of the matrix $E + zBE^*$ are within the unit circle. In Table 2, we have listed the first two decimal digits of the real and imaginary stability boundaries of the stability region of (2.7) with $E^* = VU^{-1}$ for the BK and Radau IIA correctors.

2.2.2. Region of convergence

We shall derive the region of convergence for the method (2.6) for fixed stepsizes. Let us define the stage vector iteration error

$$\varepsilon^{(j)} := \begin{pmatrix} \varepsilon_1^{(j)} \\ \varepsilon_2^{(j)} \\ \cdots \\ \varepsilon_N^{(j)} \end{pmatrix}, \qquad \varepsilon_n^{(j)} := Y_n^{(j)} - Y_n.$$

Table 2
Stability boundaries $(\beta_{\text{real}}, \beta_{\text{imag}})$ for $\{(2.7),\ E^* = VU^{-1}\}$

RK corrector	$s=2$	$s=3$	$s=4$	$s=5$
Butcher–Kuntzmann	(0.61, 0.62)	(0.49, 0.00)	(0.44, 0.00)	(0.42, 0.00)
Radau IIA	(0.92, 0.00)	(0.59, 0.61)	(0.49, 0.00)	

Subtracting (2.1) and (2.4), we find that $\varepsilon_n^{(j)}$ satisfies the linear homogeneous recursion

$$\varepsilon_n^{(j)} - E\varepsilon_{n-1}^{(j)} = zB\varepsilon_n^{(j-1)}, \quad z := \lambda h. \tag{2.8}$$

Hence, given the initial iteration error $\varepsilon^{(0)}$ and observing that $\varepsilon_0^{(j)}$ vanishes, we obtain

$$\varepsilon^{(j+1)} = zM\varepsilon^{(j)}, \quad M := L^{-1}K, \tag{2.9}$$

where L and K are the $N(s+1)$-by-$N(s+1)$ matrices

$$L := \begin{pmatrix} I & O & O & \dots & O & O \\ -E & I & O & \dots & O & O \\ . & . & . & . & . & . \\ O & O & O & \dots & -E & I \end{pmatrix}, \qquad K := \begin{pmatrix} B & O & \dots & O \\ O & B & \dots & O \\ . & . & . & . \\ O & O & \dots & B \end{pmatrix}. \tag{2.10}$$

These formulas suggest defining the *region of convergence* $\mathbb{C}$

$$\mathbb{C} := \{z: |\alpha(z)| < 1\}, \quad \alpha(z) := |z|\rho(L^{-1}K), \tag{2.11}$$

where $\rho(\cdot)$ denotes the spectral radius function. Furthermore, we observe that for any two matrices P and Q, the relation

$$\begin{pmatrix} I & O & O & \dots & O & O \\ -P & I & O & \dots & O & O \\ . & . & . & . & . & . \\ O & O & O & \dots & -P & I \end{pmatrix}^{-1} \begin{pmatrix} Q & O & \dots & O \\ O & Q & \dots & O \\ . & . & . & . \\ O & O & \dots & Q \end{pmatrix} = \begin{pmatrix} Q & O & O & O & \dots \\ PQ & Q & O & O & \dots \\ P^2Q & PQ & Q & O & \dots \\ P^3Q & P^2Q & PQ & Q & \dots \\ . & . & . & . & . \end{pmatrix} \tag{2.12}$$

holds. Applying this relation to the amplification matrix $M = L^{-1}K$ and observing that $E^i = E$, we obtain

$$M = \begin{pmatrix} B & O & O & \dots & O \\ H & B & O & \dots & O \\ H & H & B & \dots & O \\ . & . & . & . & . \\ H & H & \dots & \mathrm{H} & \mathrm{B} \end{pmatrix}, \quad H := EB = (\boldsymbol{e}\boldsymbol{b}^{\mathrm{T}}, \boldsymbol{0}). \tag{2.13}$$

Table 3
Spectral radius $\rho(A)$ for RK correctors

RK corrector	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
Butcher–Kuntzmann	0.50	0.29	0.22	0.17	0.14
Radau IIA	1.00	0.41	0.28	0.20	0.16

Notice that the matrix M is singular because the $(s+1)$st, $(2s+2)$nd, etc. columns have zero entries. This singularity can easily be removed if we redefine the error recursion (2.9) by omitting the $(s+1)$st, $(2s+2)$nd, etc. rows and columns of M, and the $(s+1)$st, $(2s+2)$nd, etc. entries of $\varepsilon^{(j)}$. Let us denote this "reduced" matrix by $\tilde{M}$. Then, it is easily verified that $\tilde{M}$ can still be represented by (2.13), provided that the matrices B and H are replaced by A and $C := \boldsymbol{e}\boldsymbol{b}^{\mathrm{T}}$, respectively. Evidently, the matrices $\tilde{M}$ and A have an equal spectral radius, which leads to the following theorem.

Theorem 2.1. *With respect to the test equation $y'(t) = \lambda y(t)$, the region of convergence of the PIRKAS GS method* (2.6) *is given by* $\mathbb{C} := \{z\colon \rho(zA) < 1\}$.

Recalling that λ is assumed to run through the spectrum of $\partial \boldsymbol{f}/\partial \boldsymbol{y}$, this theorem leads us to the convergence condition

$$h \leqslant \frac{1}{\rho(\partial \boldsymbol{f}/\partial \boldsymbol{y})\rho(A)}. \tag{2.14}$$

In Table 3, we have listed the values of $\rho(A)$ for the BK methods and Radau IIA methods with $s = 1, \ldots, 5$ (we remark that the region of convergence of the PIRKAS GS method, and therefore the condition of convergence, is the same as those of the PIRK method). Because of the relatively small values of $\rho(A)$, the stepsize restriction is not severe. A comparison with Table 2 reveals that the stability condition imposed by the predictor is considerably more severe than the convergence condition of the corrector.

The preceding considerations are "asymptotic" considerations, that is, the convergence condition is only relevant for sufficiently many iterations. In order to get insight into the convergence in the initial phase of the iteration process, we now consider the *convergence factor*. This will be the subject of the next section.

2.3. The convergence factor

The preceding considerations suggest defining the (averaged) convergence factor by the quantity

$$\alpha(N, j) := |z| \sqrt[j]{\|\tilde{M}^j\|_\infty} = \frac{|\lambda| T}{N} \sqrt[j]{\|\tilde{M}^j\|_\infty}, \tag{2.15}$$

where T denotes the length of the integration interval. First, we derive the convergence factor for $j \to \infty$ and for $N \to \infty$.

Theorem 2.2. *For any corrector* (2.1), *the convergence factor* $\alpha(N, j)$ *satisfies the relations*

$$\alpha(N, j) = |\lambda| T \frac{\rho(A)}{N} + O(j^{-1}) \quad \text{as } j \to \infty, \tag{2.16a}$$

$$\alpha(N, j) = |\lambda| T \sqrt[j]{\frac{\|\boldsymbol{b}^T\|_\infty}{j!}} + O(N^{-1}) \quad \text{as } N \to \infty. \tag{2.16b}$$

Proof. Relation (2.16a) is immediate from the asymptotic formula $\|\tilde{M}^j\|^{1/j} = \rho(\tilde{M}) + O(j^{-1}) = \rho(A) + O(j^{-1})$ as $j \to \infty$. Relation (2.16b) can be proved by an analysis of the structure of the matrices $\tilde{M}^j$. In order to get some idea of this structure, we consider the case $j = 2$. By observing that the matrix C in the lower triangle of $\tilde{M}$ is idempotent, we find

$$\tilde{M}^2 = \begin{pmatrix} A^2 & O & . & . & . \\ CA + AC & A^2 & O & . & . \\ CA + AC + C & CA + AC & A^2 & O & . \\ CA + AC + 2C & CA + AC + C & CA + AC & A^2 & O \\ . & . & . & . & . \end{pmatrix}.$$

Evidently, the maximum norm of $\tilde{M}^2$ is determined by its last row of submatrices. Hence, for any matrix A, the maximum norm of this row is given by $\|((N-2)C, (N-3)C, \ldots, 2C, C)\|_\infty + O(N)$ as $N \to \infty$. From the definition $C = \boldsymbol{e}\boldsymbol{b}^T$ it follows that

$$\|\tilde{M}^2\|_\infty = \tfrac{1}{2} N^2 \|\boldsymbol{b}^T\|_\infty + O(N) \quad \text{as } N \to \infty.$$

Since the limiting value of the norm does not depend on the matrix A, we conclude that $\|\tilde{M}^2\|_\infty = \|\tilde{M}_0^2\|_\infty + O(N)$, where $\tilde{M}_0$ is obtained from $\tilde{M}$ by replacing A with O. More generally, it can be shown that

$$\|\tilde{M}^j\|_\infty = \|\tilde{M}_0^j\|_\infty + O(N^{j-1}) \quad \text{as } N \to \infty$$

and using the relation

$$\sum_{n=1}^{N} n^q = \frac{1}{q+1} N^{q+1} + O(N^q) \quad \text{as } N \to \infty,$$

it can be shown by induction that

$$\|\tilde{M}_0^j\|_\infty = \frac{1}{j!} N^j \|\boldsymbol{b}^T\|_\infty + O(N^{j-1}) \quad \text{as } N \to \infty.$$

The result (2.16b) is now readily proved. □

It turns out that the asymptotic value for $N \to \infty$ is already reached for relatively small values of N, whereas the asymptotic value for $j \to \infty$ takes a considerable number of iterations (see Table 4 where values of $\alpha(N, j)/(|\lambda| T)$ are listed for the four-stage BK and Radau IIA correctors).

Table 4
Values of $\alpha(N, j)/(|\lambda|T)$ for the BK and Radau IIA correctors with $s = 4$

RK corrector	j	$N = 1$	$N = 2$	$N = 4$	$N = 8$	$N = 16$	$N \to \infty$
Butcher–Kuntzmann	1	0.93	0.97	0.98	0.99	1.00	1.00
	2	0.66	0.68	0.70	0.70	0.70	0.71
	4	0.42	0.44	0.44	0.45	0.45	0.45
	8	0.25	0.26	0.26	0.26	0.26	0.26
	16	0.21	0.14	0.14	0.15	0.15	0.15
	32	0.18	0.12	0.08	0.08	0.08	0.08
	$j \to \infty$	0.17	0.09	0.05	0.03	0.01	0.00
Radau IIA	1	1.00	1.00	1.00	1.00	1.00	1.00
	2	0.71	0.71	0.71	0.71	0.71	0.71
	4	0.45	0.45	0.45	0.45	0.45	0.45
	8	0.28	0.27	0.27	0.25	0.25	0.25
	16	0.24	0.16	0.15	0.15	0.15	0.15
	32	0.22	0.13	0.09	0.08	0.08	0.08
	$j \to \infty$	0.20	0.11	0.06	0.04	0.02	0.00

Finally, we consider the condition of the correction formula (2.4). Since these correction formulas couple the iterates at all step points t_n, $n = 0, 1, \ldots, N$, their condition may play a role in actual computation. We shall derive the condition of (2.4) in the case of the model equation $y' = \lambda y$. For this equation, (2.4) reduces to

$$\boldsymbol{Y}_n^{(j)} = E\boldsymbol{Y}_{n-1}^{(j)} + zB\boldsymbol{Y}_n^{(j-1)}, \quad z := \lambda h. \tag{2.4'}$$

Following the approach of Section 2.2 for the iteration errors $\varepsilon_n^{(j)}$, we drop the last component of the iterate $\boldsymbol{Y}_n^{(j)}$, for $n = 1, \ldots, N$, and we combine the "reduced" iterates $\boldsymbol{Y}_n^{(j)}$ in one vector $\boldsymbol{Y}^{(j)}$. In an analogous way as we derived (2.9), we are led to the recursion $\boldsymbol{Y}^{(j+1)} = z\tilde{M}\boldsymbol{Y}^{(j)}$, where we assumed the initial values of the IVP to be zero. Suppose that $\tilde{M}$ is perturbed by the matrix $\delta P\tilde{M}$ and $\boldsymbol{Y}^{(j)}$ by the vector $\delta Q\boldsymbol{Y}^{(j)}$, where P and Q are perturbation matrices with Q diagonal, and where δ is a small positive parameter. Then, instead of $\boldsymbol{Y}^{(j+1)}$, we obtain the perturbed iterate $\boldsymbol{Y}(\delta) = z(\tilde{M} + \delta P\tilde{M})(I + \delta Q)\boldsymbol{Y}^{(j)}$. Hence, defining the condition number $\kappa(\tilde{M}) := \|\tilde{M}\|\,\|\tilde{M}^{-1}\|$,

$$\begin{aligned}\|\boldsymbol{Y}(\delta) - \boldsymbol{Y}^{(j+1)}\| &= \delta\|_z(P\tilde{M} + \tilde{M}Q)\boldsymbol{Y}^{(j)}\| + O(z\delta^2) = \delta\|(P + \tilde{M}Q\tilde{M}^{-1})\boldsymbol{Y}^{(j+1)}\| + O(z\delta^2)\\ &\leqslant \delta(\|P\| + \kappa(\tilde{M})\|Q\|)\,\|\boldsymbol{Y}^{(j+1)}\| + O(z\delta^2).\end{aligned} \tag{2.17}$$

Thus, the magnitude of $\kappa(\tilde{M})$ estimates the effect of perturbations of $\boldsymbol{Y}^{(j)}$ on $\boldsymbol{Y}_n^{(j+1)}$. With respect to the maximum norm $\|\cdot\|_\infty$, the following result can be derived.

Theorem 2.3. *For the BK corrector the condition number* $\kappa_\infty(\tilde{M}) := \|\tilde{M}\|_\infty\,\|\tilde{M}^{-1}\|_\infty$ *is given by*

$$\kappa_\infty(\tilde{M}) = (N-1)(\|A\|_\infty + N - 1)\,\|(A^{-1}CA^{-1}, A^{-1})\|_\infty \approx N^2\|(A^{-1}CA^{-1}, A^{-1})\|_\infty. \tag{2.18a}$$

Table 5
Condition number $\kappa_\infty(\tilde{M})$ for RK correctors

RK corrector	s	$N=1$	$N=2$	$N=4$	$N=8$	$N=16$	$N\to\infty$
Butcher–Kuntzmann	2	7	38	172	728	2991	$12\,N^2$
	3	22	99	421	1738	7059	$28\,N^2$
	4	45	198	827	3377	13647	$55\,N^2$
	5	80	343	1416	5753	23192	$93\,N^2$
Radau IIA	2	7	18	36	72	144	$29\,N$
	3	18	42	84	169	337	$11\,N$
	4	34	76	153	306	611	$38\,N$

If the corrector is L-stable, then

$$\kappa_\infty(\tilde{M}) = (\|A\|_\infty + N - 1)\,\|(A^{-1}CA^{-1}, A^{-1})\|_\infty \approx N\,\|(A^{-1}CA^{-1}, A^{-1})\|_\infty. \tag{2.18b}$$

Proof. Since A is nonsingular, it can be verified that $\tilde{M}^{-1}$ is of the form

$$\tilde{M}^{-1} = \begin{pmatrix} A^{-1} & O & . & . & . & . \\ -F & A^{-1} & O & . & . & . \\ FG & -F & A^{-1} & O & . & . \\ -FG^2 & FG & -F & A^{-1} & O & . \\ . & . & . & . & . & . \end{pmatrix}, \quad F := A^{-1}\boldsymbol{e}\boldsymbol{d}^{\mathrm{T}}, \quad G := \boldsymbol{e}\boldsymbol{d}^{\mathrm{T}} - I, \quad \boldsymbol{d}^{\mathrm{T}} := \boldsymbol{b}^{\mathrm{T}}A^{-1}. \tag{2.19}$$

Using the relation $FG^j = \gamma^j F$, where $\gamma := \boldsymbol{d}^{\mathrm{T}}\boldsymbol{e} - 1$, we conclude from (2.13) and (2.19) that

$$\kappa_\infty(\tilde{M}) = (\|A\|_\infty + N - 1)\,\|Q\|_\infty, \quad Q := (\gamma^{N-2}F, \gamma^{N-3}F, \gamma^{N-4}F, \ldots, \gamma F, F, A^{-1}).$$

Hence,

$$\kappa_\infty(\tilde{M}) = (\|A\|_\infty + N - 1)\left\|\left(\frac{1-|\gamma|^{N-1}}{1-|\gamma|}F, A^{-1}\right)\right\|_\infty.$$

From the stability function $R(z)$ at infinity, that is from

$$R(z) := 1 + z\boldsymbol{b}^{\mathrm{T}}(I - zA)^{-1}\boldsymbol{e} = 1 - \boldsymbol{b}^{\mathrm{T}}A^{-1}\boldsymbol{e} + \mathrm{O}(z^{-1}) \quad \text{as } z \to \infty,$$

we see that $\gamma = \boldsymbol{b}^{\mathrm{T}}A^{-1}\boldsymbol{e} - 1 = -R(\infty)$. Hence, for BK methods we have $\gamma = (-1)^{s+1}$, and for all L-stable methods we have $\gamma = 0$. This leads us straightforwardly to the assertion of the theorem. □

This theorem shows that for large N, BK correctors possess less well-conditioned amplification matrices than Radau IIA correctors (see also Table 5), which may result in a larger *total* number of

function calls as N increases. However, from a practical point of view, it is the number of *sequential* function calls $N_{seq} = N + m$ that is important. Hence, for large N, the conditioning of the amplification matrix will not influence the *sequential* costs.

3. Implementation considerations

In an actual implementation, we are faced with aspects as the stability of the predictor formula, the number of iterations needed to reach the corrector solution, stepsize control, adapting the algorithm to a given number of processors, etc. In this section, we shall briefly discuss these issues.

3.1. The predictor

In Section 2.2, we considered the accuracy and stability of formula (2.7) for the first iterate $Y_n^{(1)}$. For larger stepsizes, this formula may lack both accuracy and stability. To circumvent this situation, we need some control on its quality. If necessary, we continue the iteration until $Y_{n-1}^{(j)}$ has the required properties to serve as a starting point to move to the next step point for computing $Y_n^{(1)}$. This can be achieved by using in (2.7) the predictor formula

$$Y_n^{(0)} = (E_n^* \otimes I) Y_{n-1}^{(j^*)}, \tag{3.1}$$

to obtain

$$Y_n^{(1)} = (E \otimes I) Y_{n-1}^{(j^*)} + h(B \otimes I) F((E_n^* \otimes I) Y_{n-1}^{(j^*)}), \quad n = 1, 2, \ldots, N, \tag{3.2}$$

where j^* is such that $Y_{n-1}^{(j^*)}$ is of sufficient quality for increasing the time index n. Thus, j^* is dynamically determined during the integration process, and, in general, j^* will depend on t_n. For the extrapolation matrix E_n^* we may choose $E_n^* = E$ (LSV predictor) or $E_n^* = VU^{-1}$ (EXP predictor, see Section 2.2.1).

3.2. Dynamic determination of j^* and m on a given number of processors

If j^* and m are dynamically determined during the integration process, then these quantities will become functions of t_n. The functions $j^*(t_n)$ and $m(t_n)$ depend on the number of processors available. In this subsection we will describe the strategy by which these functions will be determined. For clarity reasons, this description will be given for the "regular" part of the integration interval and needs slight adaptation at the start and at the end of the interval, since then fewer points (in time) are involved.

For simplicity, iterates will be indicated by $Y_n^{(j)}$, in spite of the fact that the iteration index j has different *actual* values at different time points, that is, the notation ignores that j depends on n.

Suppose that we have at our disposal a network of P processor units where each unit contains s processors (see the discussion of the computational scheme (2.6)). Then, instead of iterating on all N iterates $Y_n^{(j)}$, $n = 1, \ldots, N$, simultaneously, we shall iterate on the last P iterates $Y_\nu^{(j)}$, $\nu = n - P, \ldots, n - 1$, that do not yet have the corrector accuracy. In fact, we only proceed to the next step point if $Y_{n-P}^{(j)}$ has the corrector accuracy and if $Y_{n-1}^{(j)}$ is a safe starting point for computing

$Y_n^{(0)}$. Given a value of P, we need a criterion that signals when the time level can be increased. For that purpose, we control the correction

$$\Delta_{n-1}^{(j)} = \frac{\|(e^{\mathrm{T}}E \otimes I)(Y_{n-1}^{(j-1)} - Y_{n-1}^{(j)}\|_1}{\|(e^{\mathrm{T}}E \otimes I)Y_{n-1}^{(j-1)}\|_1}. \tag{3.3}$$

Thus, first, we require that at t_{n-P} the corrector is approximately solved by the iterate $Y_{n-P}^{(j)}$ leading to the condition

$$\Delta_{n-P}^{(j)} \leqslant \mathrm{TOL}_{\mathrm{corr}}. \tag{3.4a}$$

As soon as this condition is fulfilled, we set $m(t_{n-P}) = j$. Next, we require that the step point value at t_{n-1} is sufficiently accurate to serve as the basis for a prediction at the next time level, resulting in

$$\Delta_v^{(j)} \leqslant \mathrm{TOL}_{\mathrm{pred}}, \tag{3.4b}$$

for $v = n-1$. Since we observed that the corrections $\Delta_v^{(j)}$ are not always a monotonically increasing function of v, we imposed—as an extra safety factor—the condition that the iterates $Y_v^{(j)}$, $v = n-P+1, \ldots, n-1$ should also satisfy (3.4b). Together, these conditions determine the value of $j^*(t_{n-1})$. Notice that the dynamic PIRKAS GS method will perform like the PIRK method if $\mathrm{TOL}_{\mathrm{pred}} \to 0$.

Since the computational costs of the predictor formulas (3.1) can be ignored, the sequential costs N_{seq} of the dynamic PIRKAS GS method satisfy

$$N_{\mathrm{seq}} \leqslant \max_{1 \leqslant n \leqslant N} m(t_n) + \sum_{n=1}^{N-1} j^*(t_n).$$

Thus, the sequential costs are completely determined by the $m(t_n)$ and $j^*(t_n)$ values. Usually, N will be large with respect to $\max_n m(t_n)$, so that $\sum_n j^*(t_n)$ is the essential quantity determining the sequential costs.

3.3. Convergence of the dynamic PIRKAS GS method

In the dynamic PIRKAS GS method, the correction formula (2.4) should be adapted according to

$$\begin{aligned} Y_n^{(j)} &= (E \otimes I)Y_{n-1}^{(q(n-1,j))} + h(B \otimes I)F(Y_n^{(j-1)}), \\ &\qquad j = 1, \ldots, m(t_n);\ n = 1, 2, \ldots, N, \\ q(n, j) &:= j + j^*(t_n) - 1, \end{aligned} \tag{3.5}$$

where $Y_n^{(j)} = Y_n^{(m(t_n))}$ for $j > m(t_n)$. Notice that by setting $j^*(t_n) = 1$ and $m(t_n) = m$ for all n, we retain the recursion (2.4). The iteration error analysis of (3.5) requires the redefinition of the iteration error vectors $\varepsilon^{(j)}$. We shall illustrate this for the case where the function $j^*(n)$ is constant for all n. So, suppose that the application of the dynamic PIRKAS GS method has led to $j^*(t_n) = j^*$, j^* being a constant integer greater than 1. Then, in the (n, j) plane, the set of iterates corresponding to the points

$$(1, i + (n-1)j^* + 1), (2, i + (n-2)j^* + 1), \ldots, (n-2, i + 2j^* + 1), (n-1, i + j^* + 1), (n, i + 1)$$

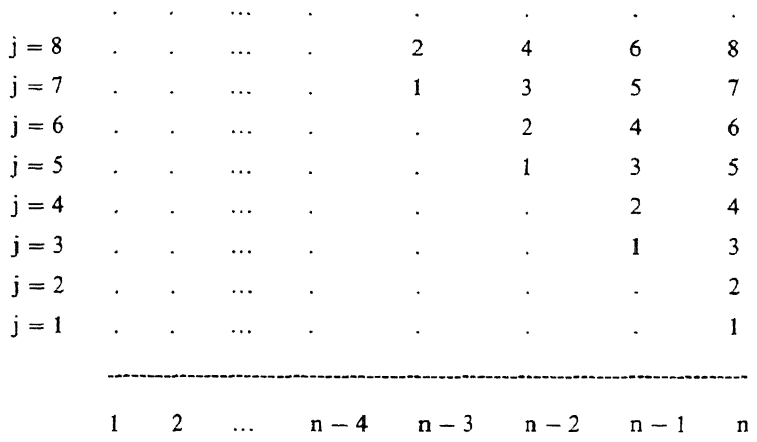

Fig. 2. Iteration index i in the case $j^* = 2$.

can be computed from the set of iterates corresponding to the points

$$(1, i + (n-1)j^*), (2, i + (n-2)j^*), \ldots, (n-2, i + 2j^*), (n-1, i + j^*), (n, i).$$

Here, i is a new iteration index assuming values $i = 1, 2, \ldots$. In Fig. 2, these sets of points are indicated by their index i for the case $j^* = 2$.

Let the iteration errors corresponding to the sets of iterates be denoted by $\eta^{(i)}$ and $\eta^{(i+1)}$, respectively. Then, it is easily verified that $\eta^{(i)}$ satisfies (2.9) with j and N replaced by i and n. Hence, with a few obvious changes, all results of Section 2 apply to (3.5), so that the convergence behaviour of the iteration errors $\varepsilon^{(j)}$ can be derived from that of the iteration errors $\eta^{(i)}$. We shall refrain from a more detailed analysis, because, as already observed in Section 3.2, the sequential costs are essentially determined by $\sum_n j^*(t_n)$, rather than by the number of iterations $m(t_n)$.

3.4. Stepsize control

In order to compare the PIRKAS GS method with results reported in the literature, we provide the method with a simple stepsize control strategy (without step rejection). A future paper will be devoted to more sophisticated stepsize control mechanisms.

An initial guess for the integration step $h_n := t_n - t_{n-1}$ can be computed by means of the standard formula (see, e.g., [9])

$$\textbf{IF } n = 1 \textbf{ THEN } \hat{h}_1 = \frac{\mathrm{TOL}}{\|f(y_0)\|_1} \textbf{ ELSE } \hat{h}_n = h_{n-1} \min\left\{2, \max\left\{\frac{1}{2}, 0.9\left(\frac{\mathrm{TOL}}{\tau_{n-1}}\right)^{1/(s+1)}\right\}\right\}, \quad (3.6a)$$

where

$$\tau_{n-1} := \|(e^{\mathrm{T}}E \otimes I)[Y^{(1)}_{n-1} - Y^{(0)}_{n-1}]\|_1, \quad n > 1 \quad (3.6b)$$

is used as an estimate for the local truncation error τ_{n-1}. Since the predictor result is of order s, this estimate is of order s, as well. In order to achieve a smooth variation of the stepsizes as a function of n, we compute a second approximation to the new integration step by applying the averaging formula

$$\textbf{IF } n = 2 \textbf{ THEN } \tilde{h}_2 = \tfrac{1}{2}(h_1 + \hat{h}_2) \textbf{ ELSE IF } n \geqslant 3 \textbf{ THEN } \tilde{h}_n = \tfrac{1}{3}(h_{n-2} + h_{n-1} + \hat{h}_n). \quad (3.6c)$$

Finally, the step $\tilde{h}_n$ is rounded to h_n such that the remaining integration interval is an integer multiple of h_n.

Notice that this stepsize strategy is rather conservative; this is due to the fact that the local truncation error is based on the difference between the prediction $Y_{n-1}^{(0)}$ (obtained by extrapolation from $Y_{n-2}^{(j^*)}$) and $Y_{n-1}^{(1)}$, the result after just one correction. This conservative error estimation is a direct consequence of the "across the steps" approach where the algorithm tries to proceed to the next step without waiting for convergence of the preceding iterate. Usually, conservative error estimates grossly overestimate the real local truncation error, resulting in rather small steps in relation to the value of TOL. As a result, this strategy tends to yield global errors that are several orders of magnitude smaller than the value of TOL. However, this is only a matter of scaling and of less practical importance. TOL still plays the role of a control parameter with the property that decreasing TOL yields a more accurate result.

4. Numerical experiments

The PIRKAS GS method {(3.1), (3.5)} described above contains as input parameters the number P of iterates that are concurrently corrected, the tolerance parameters $\mathrm{TOL}_{\mathrm{corr}}$ (for the correction at t_{n-P}) and $\mathrm{TOL}_{\mathrm{pred}}$ (for the corrections at the remaining $P-1$ points), and the tolerance parameter TOL for the stepsize. With respect to the parameter $\mathrm{TOL}_{\mathrm{corr}}$ we remark that it has been given a small value to ascertain that the corrector was more or less solved. In most experiments, the value 10^{-10} is sufficiently small; in a few situations (i.e., when the corrector is able to produce a global error less than 10^{-10}, we change to $\mathrm{TOL}_{\mathrm{corr}} = 10^{-12}$ in order not to be hampered by a too crude convergence tolerance). It may happen that the most left iterate of the block of iterates that are concurrently corrected, already satisfies the condition (3.4a) while (3.4b) is not yet satisfied. In such a situation, we do not need the corresponding processor anymore. Thus, the number of processors that is *actually needed* may change during the integration process. However, for the performance of the method it is not relevant whether we continue iterating or not.

In this section, we present a few examples illustrating the effect of the parameters P, $\mathrm{TOL}_{\mathrm{pred}}$ and TOL on the efficiency of the PIRKAS GS method. The calculations are performed using 15-digits arithmetic. The accuracy is given by the number of correct digits Δ, obtained by writing the maximum norm of the absolute error at the endpoint in the form $10^{-\Delta}$. We recall that the sequential computational complexity can be measured by N_{seq}, the total number of sequential right-hand side evaluations performed in the integration process. Furthermore, we define the average number of iterations and the average number of *sequential* iterations per step by $m^* := N^{-1}\sum_n m(t_n)$ and $m^*_{\mathrm{seq}} := N^{-1}N_{\mathrm{seq}}$.

4.1. Test problems

Widely used problems for testing nonstiff solvers are the Euler problem JACB from [9, p. 236]

$$\begin{aligned} y_1' &= y_2 y_3, & y_1(0) &= 0, \\ y_2' &= -y_1 y_3, & y_2(0) &= 1, \qquad 0 \leqslant t \leqslant 60, \\ y_3' &= -0.51 y_1 y_2, & y_3(0) &= 1, \end{aligned} \tag{4.1}$$

the Fehlberg problem (cf. [9, p. 174])

$$\begin{aligned} y_1' &= 2t y_1 \log(\max\{y_2, 10^{-3}\}), \quad y_1(0) = 1, \\ & \qquad\qquad 0 \leqslant t \leqslant 5, \\ y_2' &= -2t y_2 \log(\max\{y_1, 10^{-3}\}), \quad y_2(0) = \mathrm{e}. \end{aligned} \tag{4.2}$$

and the Lagrange problem LAGR (cf. [9, p. 237])

$$\begin{aligned} & y_j' = y_{j+10}, \quad j = 1, 2, \ldots, 10, \\ & y_{11}' = -y_1 + y_2, \\ & y_{j+10}' = (j-1) y_{j-1} - (2j-1) y_j + j y_{j+1}, \quad j = 2, 3, \ldots, 9; \qquad 0 \leqslant t \leqslant 10, \\ & y_{20}' = 9 y_9 - 19 y_{10}, \\ & y_j(0) = 0 \text{ for } j \neq 8, \quad y_8(0) = 1. \end{aligned} \tag{4.3}$$

4.2. Convergence behaviour

Since the major aim of the PIRKAS GS approach is to reduce the number of sequential iterations needed to solve the corrector, we will first present some results to illustrate the convergence behaviour. For that purpose we use the Euler problem (4.1) and we will consider the influence on the convergence when the input parameters are varied. The parameter TOL, which controls the local truncation error, is the familiar tolerance parameter occurring in any ODE code by which the accuracy of the numerical solution is controlled (see Section 3.4). Results for several values of TOL will be given in Table 7. However, choosing suitable values for the tolerance parameter $\mathrm{TOL}_{\mathrm{pred}}$ and the number of processor units P is less evident. For the 4-point BK corrector, their influence is shown in Table 6. From this table we conclude that the role of $\mathrm{TOL}_{\mathrm{pred}}$ is not very critical as long as $P \leqslant 8$. This behaviour can be explained by the fact that for small P, (3.4a) will usually be a more severe condition than (3.4b). Hence (3.4a) will force the algorithm to make several corrections to let the left point from the block that is concurrently iterated satisfy the corrector. As a consequence, the quality of all other points involved (in particular the right one which will be used to create a prediction) will be improved as well. Hence (3.4b) is then easily satisfied, even for smaller values of $\mathrm{TOL}_{\mathrm{pred}}$. For large P-values however, an iterate corresponding to a particular time level has been part of many blocks and hence many corrections have been performed at this time point. Therefore, the test (3.4a) is easily passed.

Table 6
{EXP, 4-point BK} and {EXP, 4-point Radau IIA} PC pair applied to the Euler problem (4.1) with TOL = 10^{-2}

		{EXP, 4-point BK} PC pair					{EXP, 4-point Radau IIA} PC pair				
P	TOL_{pred}	Δ	N	N_{seq}	m^*_{seq}	m^*	Δ	N	N_{seq}	m^*_{seq}	m^*
1	10^{-1}	7.4	152	1080	7.1	7.1	6.0	187	1326	7.1	7.1
	10^{-2}	7.4	152	1080	7.1	7.1	6.0	187	1326	7.1	7.1
2	10^{-1}	7.4	152	551	3.6	6.2	6.0	187	672	3.6	6.1
	10^{-2}	7.4	152	551	3.6	6.2	6.0	187	672	3.6	6.1
4	10^{-1}	7.4	153	365	2.4	8.2	6.1	189	422	2.2	7.7
	10^{-2}	7.4	153	365	2.4	8.2	6.1	189	422	2.2	7.7
8	10^{-1}	7.5	155	302	1.9	13.5	6.1	190	340	1.8	12.5
	10^{-2}	7.5	155	302	1.9	13.5	6.1	190	340	1.8	12.5
16	10^{-1}	8.3	233	381	1.6	22.2	6.6	266	414	1.6	21.6
	10^{-2}	8.0	179	327	1.8	21.2	6.4	207	356	1.7	20.9
	10^{-3}	7.4	152	301	2.0	15.7	6.0	185	367	2.0	9.2
	10^{-4}	7.4	152	414	2.7	6.6	6.0	187	537	2.9	6.1
32	10^{-1}	8.2	198	347	1.8	24.6	6.5	223	373	1.7	24.2
	10^{-2}	8.0	180	329	1.8	21.4	6.3	203	354	1.7	21.8
	10^{-3}	7.4	152	301	2.0	15.7	6.0	185	367	2.0	9.2
	10^{-4}	7.4	152	414	2.7	6.6	6.0	187	537	2.9	6.1

To guarantee that the "front" of the block is also of sufficient quality, we need a more stringent value for TOL_{pred}. From the table it is clear that crude values for this parameter result in larger truncation errors and hence an increased number of time steps. In general, we conclude that increasing P leads to an enhanced performance.

In order to see the effect of the corrector formula (2.1) on the averaged number of iterations m^*, we also listed results for the {EXP, 4-point Radau IIA} PC pair. Evidently, the BK corrector produces higher accuracies and requires less sequential function calls. Furthermore, the averaged number of iterations per step point is comparable, except for the case where a larger number of processor units is combined with a smaller value of TOL_{pred} (in the limit, the averaged number of iterations m^* approaches that of the PIRK method corresponding to $P = 1$). This difference can be explained by the particular step advance strategy used causing $j^*(t_n)$ to change discontinuously.

Confining our considerations to BK correctors, we will now test the influence of the number of stages s and the parameter TOL. Table 7 shows results for $s = 2$ and $s = 4$. In these tests we set $P = 4$ and $TOL_{pred} = 10^{-1}$. This table gives rise to the conclusion that the number of sequential calls per step is quite modest (much lower than for the PIRK method), and moreover decreases when we move to the high accuracy range. This tendency was also observed for the other problems.

4.3. Comparison with DOPRI8

Next, we will make a comparison with the code DOPRI8 (given in [9]); this code is based on the embedded RK method in [18] of order 8(7). DOPRI8 is nowadays considered as the state of the art

Table 7
{EXP, s-point BK} PC pairs with $P = 4$ applied to the Euler problem (4.1) with $\mathrm{TOL}_{\mathrm{pred}} = 10^{-1}$

s	TOL	Δ	N	N_{seq}	m^*_{seq}	m^*
2	10^{-1}	2.4	121	370	3.1	10.8
	10^{-2}	3.6	263	539	2.0	7.1
	10^{-3}	4.8	566	910	1.6	5.3
	10^{-4}	6.0	1234	1633	1.3	4.2
4	1	4.4	58	234	4.0	13.6
	10^{-1}	5.9	95	290	3.1	10.7
	10^{-2}	7.4	153	365	2.4	8.2
	10^{-3}	9.6	245	462	1.9	6.3

Table 8
Values of N_{seq} for DOPRI8 and speed-up factors for PIRKAS GS methods (with various numbers of processor units) for the Euler problem (4.1)

Code	Order	P	$\Delta = 4$	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$
DOPRI8	8	—	1083	1361	1864	2366	3038	3600	4526
PIRKAS GS	8	4	5.0	5.3	6.3	6.9	7.8	8.3	9.4
		8	5.0	5.9	7.4	8.3	8.9	8.6	8.9
		16	4.8	5.7	7.3	8.3	8.4	8.8	10.1
PIRKAS GS	10	4		6.7	8.2	9.3	10.4	10.7	11.4
		8		6.3	8.5	10.2	11.9	12.9	14.9
		16			8.1	10.0	11.8	12.5	14.2

for integrating nonstiff problems on a sequential computer. For a wide range of TOL-values, we applied DOPRI8 to the three test problems. In Tables 8–10 we present, for a number of *integer* Δ-values, the corresponding N_{seq}-values, obtained by interpolation. For the same Δ-values, we calculate the values of N_{seq} needed by the PIRKAS GS method and we list the speed-up factors with respect to DOPRI8 (defined as the quotient of the respective values of N_{seq}).

From these tables we see that the speed-up factors increase if we enter the high-accuracy region (for the PIRKAS GS method of order 10 this is of course also caused by the higher order). Furthermore, with respect to the number P, we conclude that its optimal value seems to be in the range [8, 16]. Of course, the optimal value may differ with the problem solved and also depends on the parameter $\mathrm{TOL}_{\mathrm{pred}}$. If $\mathrm{TOL}_{\mathrm{pred}}$ is chosen too large for the problem at hand, then the optimal value of P should be sufficiently small in order to prevent that condition (3.4a) is satisfied prior to (3.4b).

Table 9
Values of N_{seq} for DOPRI8 and speed-up factors for PIRKAS GS methods (with various numbers of processor units) for the Fehlberg problem (4.2)

Code	Order	P	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$	$\Delta = 11$
DOPRI8	8	—	658	824	1025	1291	1650	2033	2570
PIRKAS GS	8	4	5.6	5.7	6.0	6.5	7.0	6.9	7.3
		8	6.0	6.5	7.3	8.1	9.0	8.7	9.0
		16	5.6	6.1	6.8	7.9	7.8	8.1	9.1
PIRKAS GS	10	4	5.9	7.0	7.7	8.4	9.4	10.4	11.8
		8	6.0	7.2	8.7	10.2	12.2	13.0	14.6
		16	5.5	6.8	8.1	9.2	10.6	11.7	13.0

Table 10
Values of N_{seq} for DOPRI8 and speed-up factors for PIRKAS GS methods (with various numbers of processor units) for the Lagrange problem (4.3)

Code	Order	P	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$
DOPRI8	8	—	668	841	1161	1498	1812	2319
PIRKAS GS	8	4	3.3	3.7	4.7	5.2	5.2	5.4
		8	3.3	3.8	4.7	5.4	5.8	6.4
		16		4.5	4.9	5.2	4.9	5.0
PIRKAS GS	10	4			5.8	6.7	7.4	8.6
		8			5.6	6.9	7.6	9.1
		16				6.7	7.3	8.6

Acknowledgements

The authors are grateful to W.M. Lioen for many comments and suggestions during our discussions of the research reported in this paper. We also acknowledge the efforts of the referee who apparently made a detailed study of the paper and suggested essential improvements in the description of our PIRKAS GS method.

References

[1] A. Bellen, R. Vermiglio and M. Zennaro, Parallel ODE-solvers with stepsize control, *J. Comput. Appl. Math.* **31** (1990) 227–293.
[2] K. Burrage, The error behaviour of a general class of predictor–corrector methods, *Appl. Numer. Math.* **8** (1991) 201–216.
[3] K. Burrage, The search for the Holy Grail, or predictor–corrector methods for solving ODEIVPs, *Appl. Numer. Math.* **11** (1993) 125–141.

[4] K. Burrage, Efficient block predictor-corrector methods with a small number of iterations, *J. Comput. Appl. Math.* **45** (1993) 139–150.
[5] J.C. Butcher, On the convergence of numerical solutions to ordinary differential equations, *Math. Comput.* **20** (1966) 1–10.
[6] J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations, Runge–Kutta and General Linear Methods* (Wiley, New York, 1987).
[7] P. Chartier, Parallelism in the numerical solution of initial value problems for ODEs and DAEs, Thesis, Université de Rennes I, France, 1993.
[8] G.H. Golub and C.F. Van Loan, *Matrix Computations* (North Oxford Academic, Oxford, 1983).
[9] E. Hairer, S.P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems*, Springer Series in Comput. Math. **8** (Springer, Berlin, 2nd ed., 1993).
[10] A. Iserles and S.P. Nørsett, On the theory of parallel Runge–Kutta methods, *IMA J. Numer. Anal.* **10** (1990) 463–488.
[11] K.R. Jackson and S.P. Nørsett, Parallel Runge–Kutta methods, 1988, manuscript.
[12] K.R. Jackson, A. Kværnø and S.P. Nørsett, Order of Runge–Kutta methods when using Newton-type iteration, *SIAM J. Numer. Anal.*, to appear.
[13] K.R. Jackson and S.P. Nørsett, The potential for parallelism in Runge–Kutta methods, Part I: RK formulas in standard form, Tech. Report No. 239/90, Dept. of Computer Science, Univ. of Toronto, 1990.
[14] I. Lie, Some aspects of parallel Runge–Kutta methods, Report 3/87, Dept. of Mathematics, Univ. of Trondheim, 1987.
[15] W.L. Miranker and W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, *Math. Comput.* **21** (1967) 303–320.
[16] J. Nievergelt, Parallel methods for integrating ordinary differential equations, *Comm. ACM* **7** (1964) 731–733.
[17] S.P. Nørsett and H.H. Simonsen, Aspects of parallel Runge–Kutta methods, in: A. Bellen, C.W. Gear and E. Russo, Eds., *Numerical Methods for Ordinary Differential Equations, Proceedings L'Aquila 1987*, Lecture Notes in Math. **1386** (Springer, Berlin, 1989).
[18] P.J. Prince and J.R. Dormand, High order embedded Runge–Kutta formulae, *J. Comput. Appl. Math.* **7** (1981) 67–75.
[19] P.J. van der Houwen and B.P. Sommeijer, Parallel iteration of high-order Runge–Kutta methods with stepsize control, *J. Comput. Appl. Math.* **29** (1990) 111–127.
[20] P.J. van der Houwen and B.P. Sommeijer, Butcher–Kuntzmann methods for nonstiff problems on parallel computers, *Appl. Numer. Math.* **15** (1994) 357–374.