Pergamon

Computers Math. Applic. Vol. 30, No. 11, pp. 9–23, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00160-3

Step-Parallel Algorithms for Stiff Initial Value Problems

W. A. VAN DER VEEN
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

(Received May 1995; accepted June 1995)

Abstract—For the parallel integration of stiff initial value problems, three types of parallelism can be employed: "parallelism across the problem," "parallelism across the method" and "parallelism across the steps." Recently, methods based on Runge-Kutta schemes that use parallelism across the method have been proposed in [1,2]. These methods solve implicit Runge-Kutta schemes by means of the so-called diagonally iteration scheme and are called PDIRK methods. The experiments described in [1], show that the speedup factor of certain high-order PDIRK methods, is about 2 with respect to a good sequential code. However, a disadvantage of the high-order PDIRK methods is, that a relatively large number of iterations is needed for each step. This disadvantage can be compensated by employing step-parallelism.

Step-parallel methods are methods in which a number of steps are treated simultaneously. This form of parallelism can be applied to any predictor-corrector method. A common feature of this approach is their poor convergence behaviour, unless the various strategies are carefully designed. In this paper, we describe two strategies for the PDIRK across the steps method. Example problems tested in this paper show for the best strategy, a speed-up factor ranging from 4 to 7 with respect to the best sequential codes.

Keywords—Numerical analysis, Runge-Kutta methods, Parallelism.

1. INTRODUCTION

In the literature, several step-parallel methods for integrating stiff initial value problems of the first-order form

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y(t), f(y(t)) \in \mathrm{R}^d$$

have been proposed. Here, a step-parallel method is understood to be a method that computes concurrently solution values at different points on the t-axis. Such methods are usually based on the iterative solution of an implicit step-by-step method. The conventional approach iterates until convergence at a particular point on the t-axis before advancing to the next point on the t-axis. Step-parallel methods, however, already start the iteration process at the next point before the iteration at the preceding point has converged. In a step-parallel method we distinguish three main components:

(i) an implicit step-by-step method (the underlying corrector that we want to solve),
(ii) an iteration process (the underlying iteration scheme) that is applied at each time point, and
(iii) a strategy that determines when it is safe to advance to the next point on the t-axis, and at the same time provides an initial guess (the advancing strategy).

The research in this paper was supported by the Technology Foundation (STW) in The Netherlands.
The author wishes to thank P.J. van der Houwen and B.P. Sommeijer for their help during the preparation of this paper.

Step-parallel methods go back to Miranker and Liniger [3] in 1967 who based their method on predictor-corrector iteration of Adams-Moulton correctors. Since then, several of such methods have been proposed. For example, one of the recent step-parallel methods that has been developed is the method of Bellen and coworkers [4,5] which is based on Steffensen iteration (see also [6]).

A common feature of step-parallel methods is that they require a carefully designed advancing strategy in order to ensure convergence, and if convergent, they often require an excessive number of iterations per time point. So the challenge is to design an advancing strategy that is both efficient and reliable with respect to convergence (robustness). Our purpose is to develop a strategy that is sufficiently robust to integrate large problems arising from control engineering and circuit analysis.

The step-parallel method developed in this paper uses the 4-stage Radau IIA method as its corrector. This classical Runge-Kutta (RK) corrector has order $p = 7$ and is L-stable. For the underlying iteration process, we have chosen the Parallel Diagonal-implicit Iterated Runge-Kutta scheme (PDIRK scheme) proposed in [1]. The PDIRK scheme has a lot of intrinsic parallelism, that is, it is a method-parallel scheme. It solves the Radau IIA corrector by means of a so-called diagonal iteration process which enables parallelism across the stages. In a performance analysis given in [1], it was shown that already without step-parallelism, PDIRK based on the 4-stage Radau IIA corrector is a factor two faster than LSODE. The purpose of this paper is to decrease the effective number of iterations per point by adding an advancing strategy to obtain a step-parallel method. Consequently, we shall measure the performance in terms of these effective iterations.

In [7], we already described a first version of an advancing strategy. This first version did not include a stepsize mechanism and could only be applied to simple test problems. For a number of sufficiently simple test problems we obtained speed-up factors with respect to LSODE ranging from 4 to 7. Furthermore, in [8] we derived convergence results for the step-parallel iteration process and we proved that it has the same stability and order properties as the underlying PDIRK scheme.

In Section 2 of the present paper, we briefly describe the underlying PDIRK scheme and in Section 3, we give an exposition of the parallelism-across-the-steps mechanism. In Sections 4 and 5, we specify two advancing strategies (including stepsize mechanisms), respectively based on extrapolation of previous information and on backward differentiation formulas. Finally, in Section 6, we shall examine the performance of these advancing strategies for various, relatively difficult test problems. It turns out that the extrapolation-based advancing strategy is the most robust and efficient one yielding speed-up factors ranging from 4 to 7 with respect to LSODE.

2. A BRIEF INTRODUCTION TO THE PDIRK METHOD

The PDIRK method is a parallel method for solving the implicit Runge-Kutta corrector equations, in the case of stiff initial value problems. We shall only consider PDIRK methods that are based on the class of L-stable, stiffly accurate implicit Runge-Kutta methods. This class contains methods of arbitrarily high order.

A Runge-Kutta method approximates the solution in s points all in the interval $(t_n, t_{n+1}]$. These s point are given by

$$t_n + c_i h_{n+1}, \quad c_i \in (0,1], \qquad i = 1, \ldots, s, \quad h_{n+1} = t_{n+1} - t_n,$$

and are called stage points. The approximation in the stage point $t_n + c_i h_{n+1}$ is denoted by $Y_{n+1,i}$ and is called stage value. Using the s stage values, an approximation to the solution in the step point t_{n+1} is obtained. This step point value is denoted by y_{n+1}. In the case of stiffly accurate methods, the step point value y_{n+1} is the last stage value $Y_{n+1,s}$ ($c_s = 1$). For compact notation, the s stage values are combined in an sd-dimensional stage vector $Y_{n+1} = (Y_{n+1,i})$. For *notational convenience only*, we assume $d = 1$ in the formulas below, but in our discussion we

will take into account that we deal with nonscalar equations. In terms of the stage vector Y_{n+1}, the method is given by

$$Y_{n+1} = EY_n + h_{n+1}AF(Y_{n+1}), \qquad n = 0, 1, \ldots, N-1, \tag{1}$$

where A and E are s by s matrices and $F(Y_{n+1})$ contains the derivatives $f(Y_{n+1,i})$. Here A contains the Runge-Kutta parameters and E is given by

$$E = \begin{pmatrix} 0 & \ldots & 0 & 1 \\ \cdot & \ldots & \cdot & \cdot \\ \cdot & \ldots & \cdot & \cdot \\ 0 & \ldots & 0 & 1 \end{pmatrix}.$$

With respect to the stage vector Y_0 we remark that only the stage value $Y_{0,s}$ is needed. This stage value is given by $Y_{0,s} = y_0$.

The nonlinear equation (1) is solved by a Newton-like method,

$$Y_{n+1}^0 \quad \text{to be defined by the predictor formula,} \tag{2}$$

$$Y_{n+1}^j = Y_{n+1}^{j-1} - (I - h_{n+1}DJ_n)^{-1}\left(Y_{n+1}^{j-1} - EY_n - h_{n+1}AF\left(Y_{n+1}^{j-1}\right)\right), \tag{3}$$

$$Y_{n+1} = Y_{n+1}^m.$$

In (3) the iteration index j runs from 1 to m. In practice, m will be determined dynamically, so that it depends on n, i.e., $m = m(n)$. The matrix I is the s-dimensional identity matrix. A reasonable choice for the predictor formula is an extrapolation formula of order s. The matrix J_n represents an approximation to the Jacobian of f at y_n and the matrix D is a fixed diagonal matrix, that is chosen such that the iteration errors of the stiff components in the numerical solution are strongly damped (see [1,2], where D is chosen such that $\rho(I - D^{-1}A) \approx 0$). The iteration scheme (2),(3) arises by replacing in the modified Newton method the matrix $(I - h_{n+1}AJ_n)^{-1}$ by $(I - h_{n+1}DJ_n)^{-1}$. In [1], this type of iteration scheme was called the diagonal iteration scheme. Finally, we describe how m is determined dynamically. First we introduce the defect defined by

$$\Delta(u, v) := \sqrt{\frac{1}{d}\sum_{i=1}^{d}\left(\frac{|u_i - v_i|}{\max(|u_i|, \tau, 10^{-6})}\right)^2}, \qquad \tau = 2\frac{\text{uround}}{\text{Tol}}. \tag{4}$$

Here uround and Tol denote the unit round off and the limit for the local error estimate, respectively. The smallest value of j for which the inequality

$$\Delta\left(y_{n+1}^j, y_{n+1}^{j-1}\right) < \text{Tol}_{\text{corr}}, \qquad y_{n+1}^j = Y_{n+1,s}^j, \tag{5}$$

is satisfied, is denoted by m. The parameter Tol_{corr} is supplied by the user.

If $d > 1$, then D, E, A and J_n are replaced by the block matrices: $D \otimes I_d$, $E \otimes I_d$, $A \otimes I_d$ and $I_s \otimes J_n$. Here, $\otimes$ denotes the Kronecker product defined by $A \otimes B = (A_{ij}B)$.

Let us consider the computational aspects of the iteration scheme (2),(3). Since D is a diagonal matrix, the s components $Y_{n+1,i}^j$, $i = 1, \ldots, s$, can be computed independently from each other, so that they become available simultaneously. We shall assume that these s components are computed at the same time on s processors. This concurrent treatment of all s stage points is an example of parallelism across the method, or more specifically, parallelism across the stages. Moreover, we no longer solve a linear system of order sd, but we solve s linear systems of order d. Obviously, they can be solved simultaneously using the s processors. These stage-parallel methods are called parallel diagonally iterated Runge-Kutta (PDIRK) methods.

For PDIRK methods based on Radau IIA with $s = 1, 2, 3, 4$, it was shown in [1,2] that their application to the test equation $y' = \lambda y$ (using fixed steps) gives an iteration process that is convergent for every λ in the left half plane. Therefore, these PDIRK methods and the corresponding Radau IIA correctors have the same accuracy and stability properties, provided that $\mathrm{Tol}_{\mathrm{corr}}$ is sufficiently small. Moreover, the PDIRK methods turn out to be much cheaper than the implicit Runge-Kutta methods. This can be explained by the fact that for PDIRK the linear algebra calculations per iteration are much cheaper. Experiments reported in [1] show that PDIRK based on Radau IIA ($s = 4$) is two times more efficient than RADAU5 (the same speed-up factor was found with respect LSODE). A disadvantage of PDIRK methods is that the number of required diagonal iterations per interval is about the order of the method. Hence, for a high-order PDIRK-method (such as PDIRK based on Radau IIA with 4 stages), a relatively large number of iterations is necessary in each interval. This is where parallelism across the steps can be exploited.

3. PDIRK ACROSS THE STEPS

We shall obtain a step-parallel scheme by modifying the PDIRK iteration scheme. In the PDIRK methods, the iteration process in a point on the t-axis must be completed, before iterations are started in the next point. Instead, step-parallel methods start iterating at the next point, before the iteration process in the preceding point has been completed. An advancing strategy will determine for every point when the current iterate is good enough for providing an initial guess and to start iterating in the next step point. As soon as this happens, the iterates in these two subsequent step points are computed simultaneously. In this section, we shall describe a step-parallel method based on PDIRK. This method is called PDIRK Across the Steps (PDIRKAS). In Section 4 and 5, we shall discuss two advancing strategies.

In the following, we use the notation $I_n = (t_{n-1}, t_n]$. In the PDIRK iteration scheme (2),(3), step-parallelism cannot be used, because in order to calculate the iterates Y_{n+1}^j, $j = 0, 1, \ldots$, the finally accepted iterate $Y_n = Y_n^{m(n)}$ is needed. To enable the simultaneous computation of iterates in the intervals I_{n+1} and I_n, the iterations in the interval I_{n+1} are started as soon as the iterate in interval I_n is good enough. Let this iterate be denoted by $Y_n^{j^*(n)}$. For obtaining the corresponding step-parallel iteration scheme, we replace in (3) Y_n by $Y_n^{j^*(n)+j-1}$. The result of these changes is

$$Y_{n+1}^0 \quad \text{to be defined by the predictor formula,} \tag{6}$$

$$Y_{n+1}^j = Y_{n+1}^{j-1} - (I - h_{n+1} D J_n)^{-1} \left(Y_{n+1}^{j-1} - E Y_n^{j^*(n)+j-1} - h_{n+1} A F\left(Y_{n+1}^{j-1}\right)\right). \tag{7}$$

Here j ranges from 1 to m, where m is the smallest iteration index j for which the inequality (5) is satisfied. The iteration index $j^*(n)$ determines how many iterations must be done in the interval I_n, before the computation of the iterates in I_{n+1} is started. We have shown [8], that if j^* is independent of n, then the iteration process (6),(7) applied to the test problem $y' = \lambda y, t \in [0, T]$ converges, whenever the PDIRK iteration process (2),(3) converges. For a convergence analysis of PDIRK methods we refer to [2]. For small values of j^*, the convergence of PDIRKAS can be quite slow or there can be even initial divergence. This is partly due to the bad initial convergence behaviour in PDIRK. In view of this, j^* will be determined dynamically. Consequently, j^* depends on n.

For the predictor formula we have considered two cases: in Section 4, we discuss a predictor that is based on extrapolation of recent iteration results, i.e., $Y_{n+1}^0 = \text{Extrapolation}\left(Y_n^{j^*(n)}\right)$. Another option is to generate predictions by means of a separate stiff solver; this case will be discussed in Section 5. In both cases these predictions are almost for free. This is obvious for the extrapolation predictor, whereas the stand alone integrator can calculate its predictions on s processors concurrently with the iterations in the interval I_n.

Suppose that the iterate $Y_n^{j^*(n)}$ has just been calculated. In the next period the following iterates are computed concurrently

$$Y_n^{j^*(n)+1} = Y_n^{j^*(n)} - (I - h_n DJ_{n-1})^{-1} \left(Y_n^{j^*(n)} - EY_{n-1}^{j^*(n-1)+j^*(n)} - h_n AF\left(Y_n^{j^*(n)}\right)\right),$$

and

$$Y_{n+1}^1 = Y_{n+1}^0 - (I - h_{n+1} DJ_n)^{-1} \left(Y_{n+1}^0 - EY_n^{j^*(n)} - h_{n+1} AF\left(Y_{n+1}^0\right)\right).$$

Notice that both computations use $Y_n^{j^*(n)}$. Hereafter, for $j = 2, \ldots, m$, the iterates

$$\begin{aligned} Y_n^{j^*(n)+j} &= Y_n^{j^*(n)+j-1} - (I - h_n DJ_{n-1})^{-1} \\ &\quad \times \left(Y_n^{j^*(n)+j-1} - EY_{n-1}^{j^*(n-1)+j^*(n)+j-1} - h_n AF\left(Y_n^{j^*(n)+j-1}\right)\right), \\ Y_{n+1}^j &= Y_{n+1}^{j-1} - (I - h_{n+1} DJ_n)^{-1} \left(Y_{n+1}^{j-1} - EY_n^{j^*(n)+j-1} - h_{n+1} AF\left(Y_{n+1}^{j-1}\right)\right), \end{aligned}$$

are calculated concurrently until the iterate in I_n satisfies (5). Note, that both calculations use $Y_n^{j^*(n)+j-1}$. Similarly, $Y_n^{j^*(n)+j}$ and the iterate $Y_{n-1}^{j^*(n-1)+j^*(n)+j}$ are computed concurrently in each period. Applying this several times we see that the iterates $Y_n^{j^*(n)+j}$, $Y_{n-1}^{j^*(n-1)+j^*(n)+j}, \ldots,$ are also computed simultaneously with Y_n^j. Notice that as $j^*(n)$, $n = 1, 2, \ldots, N$, is smaller, more intervals are treated simultaneously. The average number of intervals that are being treated simultaneously depends on the number of iterations needed by PDIRK. The number of iterations per interval needed by PDIRKAS is higher than that for PDIRK. However, for PDIRKAS many iterations in an interval are done simultaneously with iterations in other intervals, resulting in significantly lower effective costs.

In the PDIRKAS iteration process each interval under treatment requires the use of s processors, for calculating the s stage values at the same time. If in an interval, the current iterate satisfies (5), then the s processors corresponding to that interval are assigned to the first interval at the right where the iteration process has not been started yet. Note, that PDIRKAS uses both parallelism across the stages and parallelism across the steps.

The choice of the mechanism for determining $j^*(n)$ and the predictor formula compose the advancing strategy. Furthermore, these choices determine whether PDIRKAS is robust and efficient. For instance, if $j^*(n)$ is small, then PDIRKAS may become divergent. This can be due to bad initial guesses. Moreover, the stepsize mechanism will be bad if it uses the initial guesses. If the predictor does not depend on $Y_n^{j^*(n)}$ or if the initial guess is good then divergence can still occur in PDIRKAS by the amplification of iteration errors as was shown in [8]. Nevertheless, a lot of step-parallelism is used. On the other hand, if $j^*(n)$ is relatively large, then we have a robust method, using step-parallelism only modestly. In order to develop efficient step-parallel methods the underlying strategy must be designed carefully.

The predictor to be discussed in Section 4 uses $Y_n^{j^*(n)}$. An appropriate advancing strategy has to ensure fast convergence of the PDIRKAS iteration process as well as to yield a good, high-order local error estimate (the corrector we solve is an high-order method). In this case the iteration index $j^*(n)$ will be the smallest j for which the iterate Y_n^j is sufficiently accurate. The predictor to be discussed in Section 5 is given by a stand-alone stiff ODE solver. Here the main purpose is to ensure a good convergence of the PDIRKAS iteration process. In the last case, the initial guess Y_{n+1}^0 no longer depends on iterates in the interval I_n. So, we have much freedom in choosing a criterion for $j^*(n)$. For instance, $j^*(n)$ can be based on the iteration process in interval I_{n-k}, with k a small positive integer.

We have selected the following two predictor formulas:

- Y_{n+1}^0 is the extrapolation of order s using $Y_n^{j^*(n)}$.
- Y_{n+1}^0 is the result of the application in the stage points of the 2-step Backward Differentiation Formula (BDF) using EY_n^0 and EY_{n-1}^0. The computation of Y_{n+1}^0 is done concurrently with the first $j^*(n)$ iterations in interval I_n.

With these two choices for the predictor, along with their definitions of $j^*(n)$, we have two PDIRKAS strategies. In the next two sections we will give the complete description of these two strategies.

In this paper, we shall restrict our considerations to the *computational complexity* of the method on a parallel computer. Communication issues will be subject of a future paper. The computational complexity will be referred to as "the effective costs", and will be expressed in terms of d-dimensional diagonal iterations (see (3)). In calculating the effective costs, all d-dimensional iterations that can be done simultaneously are counted as one. In particular, the effective costs of computing $Y_{n+1}^j, Y_n^{j^*(n)+j}, Y_{n-1}^{j^*(n-1)+j^*(n)+j}, \ldots$, is just one unit of effective costs. For measuring the effective costs we have run an implementation of our step-parallel method on a sequential computer while keeping track of the computational complexity as if it had been executed on a parallel computer. In a forthcoming paper we shall report on the performance of an actual implementation of our step-parallel method on a parallel computer, including communication effects.

Having described the step-parallel method, we shall discuss what type of parallel computer is most suitable for implementing PDIRKAS. We can exploit two kinds of parallelism: parallelism across the stages, and across the steps. For parallelism across the stages, s processors are needed to compute in every iteration step $Y_{n+1,i}^j, i = 1, \ldots, s$. After each iteration, the new stage values must be broadcasted to the other $s-1$ processors. Because of the many communications, a shared memory system is appropriate. For using step-parallelism, we can employ a cluster of such shared memory systems. In this type of parallelism, each system has to communicate information to only one other system.

4. PDIRKAS USING THE EXTRAPOLATION PREDICTOR

In this section, we describe the strategy for the PDIRKAS method, that uses for the predictor the extrapolation formula of order s. This strategy will be referred to as PDIRKAS(EXT). First, we shall give the predictor formula, followed by the mechanism for determining $j^*(n)$.

The initial guess Y_{n+1}^0 is given for $n \geq 1$ by the extrapolation formula of order s

$$Y_{n+1}^0 = E_{n+1} Y_n^{j^*(n)},$$

where E_{n+1} satisfies the order conditions

$$E_{n+1}(c-e)^k = (r_n c)^k, \qquad r_n = \frac{h_{n+1}}{h_n}, \qquad k = 0, 1, \ldots, s-1.$$

Here $c = (c_1, \ldots, c_s)^T$ and e is the s-dimensional vector with unit entries. This gives

$$E_{n+1} = VU^{-1}, \qquad U := \left(e, c-e, \ldots, (c-e)^{s-1}\right), \qquad V := \left(e, r_n c, \ldots, (r_n c)^{s-1}\right).$$

To calculate Y_{n+1}^0, the steplength h_{n+1} and $j^*(n)$ are needed. Having given the predictor formula for Y_{n+1}^0, there remains the mechanism for determining j^* and h_{n+1}.

First, we shall describe how $j^*(n)$ is determined using the iterates in the interval I_n and in the previous intervals. Because the iteration process in interval I_1 is a PDIRK iteration process, the definition for $j^*(1)$ differs from the general case. Unless mentioned otherwise, we assume that $n \geq 2$. Since the initial guess Y_{n+1}^0 depends on $Y_n^{j^*(n)}$, the iteration index $j^*(n)$ will be the smallest value of j for which Y_n^j is "sufficiently accurate". More precisely, the iteration index $j^*(n)$ is the smallest value of j, for which Y_n^j satisfies a number of criteria. The first criterion is that Y_n^j approximately solves equation (1) and reads

$$\mathrm{res}(Y_n^j, n, j) < p_{\mathrm{abs}} \mathrm{Tol},$$

with p_{abs} a positive parameter. Here, $\text{res}(Y_n^j, n, j)$ denotes the residue of Y_n^j, with respect to (1) at time step n and iteration level j and is given by

$$\text{res}(B, n, j) = \Delta\left(e_s^\top B, e_s^\top Y_{n-1}^{j+j^*(n-1)} + h_n e_s^\top AF(B)\right).$$

Here, the defect $\Delta(\cdot,\cdot)$ is given by (4) and B is some approximation for Y_n that is computed simultaneously with $Y_n^{j+j^*(n-1)}$. We need an additional criterion, because the first one does not lead to good local error estimates.

For the choice of the second criterion we make use of the following observations. It is very possible that the initial iterates in an interval are converging too slowly, or that there is a slight initial growth of the iteration error. This last phenomenon already occurs for the test problem $y' = \lambda y$ for certain values of λ lying in the left half plane [7,8]. Therefore, in the beginning of the iteration process in interval I_n, the information in the interval I_{n-1}, (e.g., $Y_{n-1}^{j^*(n-1)+j}$) is much more reliable than information in the interval I_n (e.g., Y_n^j).

In order to decide whether Y_n^j is good enough, we compare it with an alternative approximation for Y_n. Considering the observation just given, a suitable alternative (or reference) approximation to Y_n is provided by a very cheap separate method, that only uses the *most recent* information in the interval I_{n-1}. We have taken as a reference solution the s^{th} order extrapolation of the iterate in the interval I_{n-1} that is calculated simultaneously with Y_n^j. This updated initial guess for Y_n will be denoted by G_n^j, and is defined by $E_n Y_{n-1}^{j^*(n-1)+j}$, and will be computed for $j = 1, \ldots, j^*(n)$.

As long as Y_n^j yields a larger residue than G_n^j, the iterate Y_n^j is not sufficiently accurate and the iteration process in I_{n+1} is not started. So the second criterion is given by

$$\text{res}(Y_n^j, n, j) < p_{\text{rel}} \text{res}(G_n^j, n, j),$$

where $p_{\text{rel}} \in (0,1)$ and $\text{res}(G_n^j, n, j)$ denotes the residue of G_n^j, given by

$$\text{res}(G_n^j, n, j) = \Delta\left(e_s^\top G_n^j, e_s^\top Y_{n-1}^{j+j^*(n-1)} + h_n e_s^\top AF\left(G_n^j\right)\right).$$

In conclusion, we take $j^*(n)$ to be the smallest iteration index j satisfying

$$\text{res}(Y_n^j, n, j) < \gamma \min\left(p_{\text{rel}} \text{res}(G_n^j, n, j), p_{\text{abs}} \text{Tol}\right), \tag{8}$$

with $\gamma = 1$ and as a precaution we impose in the interval I_{n-1} a similar condition with $\gamma = 0.5$.

There are situations where it takes a lot of iterations to satisfy these criteria, while the convergence is good. This happens, for instance, if the defect $\Delta(y^j, y^{j-1})$ is small and $\text{res}(Y_n^j, n, j)$ is large. To deal with these cases, Y_n^j is also considered to be sufficiently accurate if the defect $\Delta(y_n^j, y_n^{j-1})$ is less than $\min(10^{-5}, 10^{-3}\text{Tol})$.

The role of the parameters p_{rel} and p_{abs} is discussed below.

If $n = 1$, then we take $Y_{n,i}^0 = y_0$ for $i = 1, \ldots, s$, and we define $j^*(1)$ to be the smallest value of $j \geq 2$ for which

$$\Delta(y_1^j, y_1^{j-1}) < \text{Tol}_1$$

holds. Here Tol_1 is a method parameter with default value 10^{-4}. From now on we assume that $n \geq 1$.

Let us consider step rejection in PDIRKAS(EXT). Although several intervals are treated simultaneously, we shall only reject steps in that interval, where the iteration index j does not exceed j^*. Assume that this is the interval I_n. The step h_n is rejected if either the local error estimate is larger than Tol or if the convergence is too slow. If the step is accepted then the iteration process in interval I_{n+1} is started, and this is the step that can be rejected. The local error estimate is only calculated when the iterate is sufficiently accurate. Therefore, step rejection due to a too large local error can only occur for $j = j^*(n)$. On the other hand, it may happen that there is

slow convergence. To avoid this, we shall halve the step in the interval I_n when at least one of the following conditions is violated in the interval I_n:

- $j^*(n) \leq \max j^*$,
- $\mathrm{res}(Y_n^j, n, j) < \mathrm{res\,lim}$, for $j > j\mathrm{conv}$,
- $\Delta(y_n^j, y_n^{j-1}) < 1$, for $j \geq 2$.

Here j assumes all values for which Y_n^j is not sufficiently accurate. Furthermore, $\max j^*$, res lim and $j$conv are method parameters (see Section 6 for their values). In view of the possible initial growth, the integer valued parameter $j$conv should not be too small.

Finally, we describe how h_{n+1} is obtained. As an indicator for the behaviour of the local error in the corrector we take

$$\mathrm{err}_n = \begin{cases} \Delta\left(y_n^{j^*(n)}, e_s^T G_n^{j^*(n)}\right) & \text{if } n > 1 \\ \Delta\left(y_1^{j^*(1)}, y_1^1\right) & \text{if } n = 1, \end{cases}$$

which is of order s. If $\mathrm{err}_n < \mathrm{Tol}$, then the step is accepted and h_{n+1} is given by

$$h_{n+1} = \frac{h_n}{\max\left(0.6, \min\left(3.0, (1/0.8)\sqrt[s]{(\mathrm{err}_n/\mathrm{Tol})}\right)\right)}. \tag{9}$$

Conversely, if $\mathrm{err}_n \geq \mathrm{Tol}$ then the step is rejected and (9) is used as the new steplength. Having described PDIRKAS(EXT) we discuss the role of p_{rel} and p_{abs}. These parameters determine $j^*(n)$ and, therefore, the stepsize and the convergence of the PDIRKAS iteration process. For small values of p_{rel} and p_{abs}, $j^*(n)$ will be relatively large. Consequently, the local error estimator is of good quality resulting in a relatively small number of steps. However, less intervals are treated simultaneously. Hence, small values of the parameters leads to inefficient strategies. On the other hand, if these two values are large and, therefore, causing $j^*(n)$ to be small, PDIRKAS may become divergent, because the initial guesses steadily deteriorate. Furthermore, the local error estimate gets worse as the parameters become larger, with the effect that more steps are needed to achieve a certain accuracy. However, the amount of step-parallelism is relatively high. So there are optimal values of the two parameters, that give the required accuracy at a minimal effective cost. Experiments show that the performance is not sensitive to small changes in the two parameters. Moreover, the optimal values are more or less problem independent. In our implementation with the Radau IIA corrector, we have taken $p_{\mathrm{abs}} = p_{\mathrm{rel}} = 0.5$.

Experiments show that the number of intervals that are treated concurrently may become large (up to 30) temporarily. However, most of the time the number of intervals under concurrent treatment is only a fraction of this. We will describe a bound K for this number. As a consequence, the conditions (8) may be satisfied while the number of processors in use equals the number K. This forces the method to continue the iteration, thus increasing $j^*(n)$. The resulting PDIRKAS(EXT) algorithm is denoted by PDIRKAS(EXT,K). Only for small K (say between 1 and 8) this restriction alters the value of j^* significantly.

Next, consider the effective costs. A straightforward implementation on a parallel computer yields an effective cost of $j^*(n) + 1$ units in the interval I_n. Assume that $Y_{n-1}^{j^*(n-1)}$ and Y_n^0 have just been calculated. First, the iterates $Y_n^j, j = 1, \ldots, j^*(n)$ are computed. When this has been done, the PDIRKAS method has to verify that the iterate $Y_n^{j^*(n)}$ satisfies (8). For this verification we need $F\left(Y_n^{j^*(n)}\right)$ and $F\left(G_n^{j^*(n)}\right)$. After these function evaluations, we advance to the interval I_{n+1} and compute Y_{n+1}^0. Hence, $F\left(Y_n^{j^*(n)}\right)$ is calculated before $F\left(Y_{n+1}^0\right)$ can be calculated. Because $F\left(Y_n^{j^*(n)}\right)$ is the first part of the computations for $Y_n^{j^*(n)+1}$, a substantial part of the calculation of $Y_n^{j^*(n)+1}$ is completed, before Y_{n+1}^1 can be computed.

However, we can reduce the effective costs in I_n to $j^*(n)$, as follows. Assume that the iterations have already been started in interval I_n, while the iterations in interval I_{n+1} have not been initiated yet. When an iterate Y_n^j in I_n has just been computed, we act as if this iterate is accurate enough in order to advance to the interval I_{n+1}. Therefore, we calculate $Y_{n+1}^0 = E_{n+1}Y_n^j$ and $F\left(Y_{n+1}^0\right)$ simultaneously with $F\left(Y_n^j\right)$. Only when Y_n^j satisfies condition (8), we really advance the iteration process to interval I_{n+1}, otherwise we just ignore $F\left(Y_{n+1}^0\right)$ and compute a new Y_{n+1}^0 based on Y_n^{j+1} and repeat the procedure just described once more. These additional calculations require s additional processors. In this fashion, $F\left(Y_n^{j^*(n)}\right)$ and $F\left(Y_{n+1}^0\right) = F\left(E_{n+1}Y_n^{j^*(n)}\right)$ are calculated simultaneously. Because these function evaluations are the first parts of the calculation of $Y_n^{j^*(n)+1}$ and Y_{n+1}^1, these iterates are also computed simultaneously (E_{n+1} is almost for free). In view of this, the effective costs in interval I_n are $j^*(n)$ units.

The total effective costs are given by $\sum_{n=1}^{N-1} j^*(n)+m(N)$ plus the effective costs of all iterations carried out in rejected steps. The number of processors needed by PDIRKAS(EXT,K) equals $(K+2)s$. Here, $2s$ processors are used for computing $F\left(Y_{n+1}^0\right) = F\left(E_{n+1}Y_n^j\right)$ and $F\left(G_n^j\right)$ simultaneously with $F\left(Y_n^j\right)$.

5. PDIRKAS USING THE BACKWARD DIFFERENTIATION FORMULA

We have implemented several strategies using BDF, the best of which will be presented here. This strategy uses for the initial guess Y_{n+1}^0 the L-stable, two-step BDF and we shall refer to it as the PDIRKAS(BDF) strategy. This predictor has to yield an initial guess for $Y_{n+1,i}$ in every stage point. These initial guesses are calculated concurrently on s processors. The implicit BDF equations are solved using the modified Newton method. This method is stopped as soon as the defect (4) between two subsequent iterations is less than $\min(10^{-5}, 10^{-3}\text{Tol})$. Here Tol is the upper bound for the local error estimate. If after 5 iterations this criterion is not satisfied, then the step is halved and new BDF approximations are calculated.

The local error, which is only controlled in the step points, is given by the defect (4), where u and v correspond to the approximations obtained by the two-step and three-step BDF and is denoted by err_n. This local error estimate is of order 3. The three-step BDF approximation is computed concurrently with the other s BDF approximations. The step h_n is accepted if $\text{err}_n < \text{Tol}$. In that case, the initial guess for the steplength is given by

$$h_{n+1} = \frac{h_n}{\max\left(0.66, \min\left(5.0, (1/0.8)\sqrt[3]{(\text{err}_n/\text{Tol})}\right)\right)}.$$

Otherwise, the step is rejected and the new steplength for h_n is given by the right hand side of the preceding formula.

We have taken as definition for $j^*(n)$: $j^*(n)$ is the smallest iteration index j of Y_n such that the residue of the iterate in interval I_{n-k}, that is computed concurrently with Y_n^j, is a factor a_k smaller than $\text{res}\left(Y_{n-k}^0, n-k, 0\right)$, i.e.,

$$\text{res}\left(Y_{n-k}^{q(j)}, n-k, q(j)\right) < a_k \text{res}\left(Y_{n-k}^0, n-k, 0\right).$$

Here $Y_{n-k}^{q(j)}$ denotes the iterate in interval I_{n-k} that is computed simultaneously with Y_n^j, and k is a small positive integer. In addition we require that:

$$\Delta\left(y_n^j, y_n^{j-1}\right) < 0.1,$$

and

$$\text{res}\left(Y_n^j, n, j\right) < 0.01 \quad \text{or} \quad \Delta\left(y_n^j, y_n^{j-1}\right) < 0.01.$$

These two last criteria prevent the propagation of instabilities. For $n \leq k$ the value of $j^*(n)$ is the smallest value of j for which

$$\Delta\left(y_n^j, y_n^{j-1}\right) < \mathrm{Tol}_1,$$

with $\mathrm{Tol}_1 = 10^{-4}$. Optimal values of the parameters k and a_k have to be determined experimentally. Experiments show that they are more or less problem-independent. We use the parameter values $k = 3$ and $a_k = 0.01$.

As in the PDIRKAS(EXT) case we shall describe a bound K for the number of intervals that are treated concurrently. The resulting method is denoted by PDIRKAS(BDF,K). An apparent disadvantage of this approach is that the local error estimate is independent of the number of stages of the corrector and, consequently, independent of its order.

In the PDIRKAS(BDF) process it happens that Y_{n+1}^1 has to be be calculated, while the calculations for Y_{n+1}^0 are not completed yet. This situation rarely arises because the maximal number of BDF iterations is limited to 5 (the average number of iterations per point turns out to be between 2 and 3).

6. PERFORMANCE EVALUATION OF PDIRKAS

6.1. Numerical Experiments

In our experiments we use the four-stage Radau IIA method as the underlying corrector. Since we shall iterate this corrector until convergence, PDIRKAS has the same order and stability properties, that is, it has step point order 7, stage order 4 and it is L-stable.

We distinguish four types of parameters:

(i) problem parameters like initial values, integration interval, etc., to be specified in Section 6.2,
(ii) input parameters to monitor the integration process and to be specified by the user,
(iii) strategy parameters that are part of the code, and
(iv) output parameters, that will be specified in Section 6.3.

The input parameters are Tol, $\mathrm{Tol}_{\mathrm{corr}}$, K and h_0. Tol is the upper bound for the local error estimate, and $\mathrm{Tol}_{\mathrm{corr}}$ determines when the iteration process in the successive intervals can be terminated (cf. (5)). Since a relatively small value of $\mathrm{Tol}_{\mathrm{corr}}$ only slightly increases the number of intervals treated simultaneously, while the effective costs remain approximately the same, we have chosen $\mathrm{Tol}_{\mathrm{corr}} = 10^{-12}$, unless mentioned otherwise. Furthermore, K is the maximum number of intervals that the user allows to be treated simultaneously, and h_0 is the initial stepsize.

PDIRKAS(EXT) contains the strategy parameters $\max j^*$, $j\mathrm{conv}$, and $\mathrm{res\,lim}$, which are respectively chosen 20, 7 and 0.1. The strategy parameters for PDIRKAS(BDF) are given in Section 5.

For the calculations, 15-digits arithmetic was used. For a number of test problems we shall give results obtained by PDIRKAS(EXT) and PDIRKAS(BDF). In order to appreciate these results, we compare them with PSODE. PSODE (Parallel Software for ODEs) has been developed in [9] and is, like PDIRKAS, based on PDIRK iteration of the four-stage Radau IIA corrector. This facilitates an easy mutual comparison in terms of effective numbers of diagonal iterations.

Finally, we remark that in both PDIRKAS strategies we have refrained from introducing a mechanism for updating the Jacobian. Since our present implementation of PDIRKAS updates the Jacobian in each step, PSODE was modified accordingly.

6.2. Test Problems

The first test problem is the electric ring modulator [10], which contains 15 differential equations. Some of them are highly nonlinear. This set of equations contains a parameter C_s, by which a DAE or ODE can be realized. We have chosen $C_s = 10^{-9}$, resulting in a stiff ODE.

The second test problem is the Robertson kinetics example, which originates from chemistry:

$$\begin{aligned} \frac{dy_1}{dt} &= -0.04y_1 + 10^4 y_2 y_3 \\ \frac{dy_2}{dt} &= 0.04y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2 \\ \frac{dy_3}{dt} &= 3 \cdot 10^7 y_2^2 \\ y(0) &= (1,0,0)^\top \end{aligned} \tag{10}$$

and with $t \in [0, 10^8]$.

We also include two van der Pol equations:

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= 50(1 - y_1^2)y_2 - y_1 \\ y(0) &= (2,0)^\top \end{aligned} \tag{11}$$

on [0,83] as the third test problem and

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= \left((1 - y_1^2)y_2 - y_1\right) 10^6 \\ y(0) &= (2,-0.66)^\top \end{aligned} \tag{12}$$

on [0,2] as the fourth. These ODEs have changes in their components in an almost discontinuous way (especially (12)).

Our fifth test problem is the linear Prothero-Robertson problem:

$$\begin{aligned} \frac{dy_1}{dt} &= -\frac{1}{\varepsilon}\left(y_1 - \cos(y_2)\right) - \sin(y_2) \\ \frac{dy_2}{dt} &= 1 \\ y(0) &= (1,0)^\top \end{aligned} \tag{13}$$

on [0,10] and with $\varepsilon = 10^{-3}$. The exact solution is given by $y(t) = \cos(t)$.

The last one is the electric inverter [11]:

$$\begin{aligned} \frac{dy_i}{dt} &= \frac{5 - y_i}{RC} - \frac{K}{C} g(y_{i-1}, y_i), \qquad i = 1, \ldots, 4, \\ R &= 5000, \qquad C = 0.2 \cdot 10^{-12}, \qquad K = 2 \cdot 10^{-4}, \\ g(u,v) &= (\max(u-1,0))^2 - (\max(u-v,0))^2, \\ y(0) &= (5, 0.5, 5, 0.5)^\top, \end{aligned} \tag{14}$$

$$y_0(t) = \begin{cases} 0 & \text{if } t \le 0.5 \cdot 10^{-8} \vee t \ge 1.75 \cdot 10^{-8} \\ 10^9 t - 5 & \text{if } t \in [0.5 \cdot 10^{-8}, 1 \cdot 10^{-8}] \\ 5 & \text{if } t \in [1 \cdot 10^{-8}, 1.5 \cdot 10^{-8}] \\ -2 \cdot 10^9 t + 35 & \text{if } t \in [1.5 \cdot 10^{-8}, 1.75 \cdot 10^{-8}] \end{cases}$$

on $[0, 2.5 \cdot 10^{-8}]$.

6.3. Numerical Results

For the experiments we recorded the following quantities:

- Tol: the upper bound for the local error estimate.
- N: the number of accepted steps.
- nsd: the relative accuracy in significant digits of the approximation in the endpoint, given by the minimum of

$$-^{10}\log \frac{|y_i^{ex} - y_i^{\mathrm{app}}|}{\max(|y_i^{ex}|, 10^{-6})},$$

  where i runs from 1 to d. Here y^{ex} and y^{app} respectively denote the exact solution and its approximation in the endpoint. Components with absolute values smaller than 10^{-6} are treated differently because this is also done in the defect (4).
- $K_{\max}$: the maximal number of intervals that are treated simultaneously.
- K_{av}: the average number of intervals that are treated simultaneously.
- C_{eff}: effective costs, the number of diagonal iterations (including iterations in rejected steps). Here all diagonal iterations, that can be done concurrently are counted as one unit.
- j^*_{av}: the average value of j^*.
- N_{reject}: the total number of rejected steps (due to convergence failure or local error control).
- m_{av}: the average number of iterations performed in an interval (including iterations done in a step rejection).

First, we shall consider how the parameter K in the PDIRKAS(EXT,K) method affects the performance. Because the maximal number of intervals that are treated concurrently is at most K, unnecessary continuation of the iteration process should be avoided for small K-values. Therefore, we have used here $\mathrm{Tol}_{\mathrm{corr}} = 10^{-9}$. In Table 1 (see appendix), the influence of K is shown for the first test problem. For this small value of $\mathrm{Tol}_{\mathrm{corr}}$, PDIRKAS(EXT,2) is about two times cheaper than PDIRKAS(EXT,1). Comparing the average number of iteration per step, given by m_{av}, we see that in an interval the PDIRKAS(EXT,2) iteration process closely resembles the PDIRK-iteration process. For larger values of K the performance does not get better any more and becomes more or less independent of K.

In Tables 2 to 7, (see appendix) we give the results of PDIRKAS(EXT) and PDIRKAS(BDF) when applied to the various test examples; for evaluating the performance we give the results obtained with PSODE as well. For PDIRKAS(EXT,4) we used $\mathrm{Tol}_{\mathrm{corr}} = 10^{-9}$ instead of $\mathrm{Tol}_{\mathrm{corr}} = 10^{-12}$. As can be seen from these tables, PDIRKAS(EXT,4) is almost as good as PDIRKAS(EXT,10). If the parameter $\mathrm{Tol}_{\mathrm{corr}}$ used in PDIRKAS(EXT,4) is smaller than 10^{-9}, this is no longer true. Comparing PDIRKAS(EXT,10) and PDIRKAS(EXT,30), it turns out that the performance of the stepsize mechanism in PDIRKAS(EXT,10) is slightly better than that of PDIRKAS(EXT,30), because of a better convergence behaviour (see m_{av}). Assuming that there are sufficiently many processors, PDIRKAS(EXT,10) is the best of the three PDIRKAS(EXT) methods.

For PDIRKAS(BDF), the experiments show that PDIRKAS(BDF,4) is slightly less efficient than PDIRKAS(BDF,10). From the tables it is apparent that PDIRKAS(BDF,30) is better than PDIRKAS(BDF,10), although the differences are small. Therefore, taking into account the large number of extra processors needed, PDIRKAS(BDF,10) is to be preferred. With respect to the van der Pol equations (11),(12), we remark that PDIRKAS(BDF) can not handle this problem, because the order of accuracy of BDF is too low.

6.4. Comparison of PDIRKAS(EXT) and PDIRKAS(BDF)

Comparing PDIRKAS(EXT,10) and PDIRKAS(BDF,10), we conclude that the first method is more efficient and more reliable than PDIRKAS(BDF,10). Comparing PDIRKAS(EXT,10) with

PSODE shows for a broad class of test problems that the speed-up factor ranges from 2 to 3.5. Recall that PSODE is twice as efficient as LSODE. Consequently, PDIRKAS(EXT,10) is 4 to 7 times more efficient than LSODE.

REFERENCES

1. P.J. van der Houwen and B.P. Sommeijer, Iterated Runge-Kutta methods on parallel computers, *SIAM J. Sci. Stat. Comput.* **12**, 1000–1028 (1991).
2. P.J. van der Houwen and B.P. Sommeijer, Analysis of parallel diagonally implicit iteration of Runge-Kutta methods, *APNUM* **11**, 169–188 (1993).
3. W.L. Miranker and W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, *Math. Comp.* **21**, 303–320 (1967).
4. A. Bellen, R. Vermiglio and M. Zennaro, Parallel ODE-solvers with stepsize control, *JCAM* **31**, 277–293 (1990).
5. A. Bellen, Parallelism across the steps for difference and differential equations, In *Lecture Notes in Mathematics*, p. 1386, Springer-Verlag, (1987).
6. P. Chartier, Parallelism in the numerical solutions of initial value problems for ODEs and DAEs, Thesis, Université de Rennes I, France, (1993).
7. P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen, Parallelism across the steps in iterated Runge-Kutta methods for stiff initial value problems, *Numerical Algorithms* **8**, 293–312 (1994).
8. W.A. van der Veen, J.J.B. de Swart and P.J. van der Houwen, Convergence aspects of step-parallel iteration of Runge-Kutta methods, *APNUM* (1995) (to appear).
9. B.P. Sommeijer, Parallel-iterated Runge-Kutta methods for stiff ordinary differential equations, *JCAM* **45**, 151–168 (1993).
10. E. Hairer, C. Lubich and M. Roche, The numerical solution of differential-algebraic systems by Runge-Kutta methods, *Lecture Notes in Mathematics*, p. 1409, Springer-Verlag, (1989).
11. W. Kampowski, P. Rentrop and W. Schmidt, *Classification and Numerical Simulation of Electric Circuits*, Math. Inst. Tech., Univ. Munchen, (1991).

APPENDIX

Table 1. Results for the Ring modulator using PDIRKAS(EXT,K), with K=1,2,4, 8,10 and $\mathrm{Tol}_{\mathrm{corr}} = 10^{-9}$.

K	Tol	N	nsd	$K_{\max}(K_{\mathrm{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
1	0.01	3174	5.8	1	27561	6.7	788	8.5
2	0.01	3166	5.9	2(1.7)	14287	3.5	760	8.8
4	0.01	3167	5.9	4(3.4)	10825	2.7	765	12.7
8	0.01	3190	5.9	8(4.6)	10278	2.6	741	15.9
10	0.01	3199	5.8	10(4.8)	10245	2.5	765	16.2

Table 2. Results for the Ring modulator.

Method	Tol	N	nsd	$K_{\max}(K_{\mathrm{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)[1]	0.01	3167	5.9	4(3.4)	10825	2.7	765	12.7
	0.002	4647	6.7	4(3.3)	15223	2.8	798	11.9
(EXT,10)	0.01	3204	5.9	10(6.7)	10443	2.6	738	22.7
	0.002	4707	6.7	10(6.6)	15062	2.7	778	22.0
(EXT,30)	0.01	3214	5.9	30(7.1)	10275	2.5	749	23.8
	0.002	4750	6.7	23(7.0)	15128	2.7	868	23.3
(BDF,10)	0.01	3556	2.9	10(8.1)	11092	3.1	1112	26.2
	0.005	4766	3.4	10(8.3)	14762	3.1	1333	26.4
PSODE	10^{-4}	1978	4.3	1	12818		453	6.5
	10^{-5}	3017	5.7	1	18655		675	6.2
	10^{-6}	4671	7.2	1	28438		974	6.1

[1](EXT,4) always uses $\mathrm{Tol}_{\mathrm{corr}} = 10^{-9}$

Table 3. Results for the Robertson kinetics example (10).

Method	Tol	N	nsd	$K_{\max}(K_{\text{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)	0.1	91	7.3	4(2.0)	374	4.1	0	9.2
	0.01	127	7.3	4(2.2)	438	3.5	0	8.7
(EXT,10)	0.1	93	7.3	10(3.3)	381	4.1	1	14.4
	0.01	128	7.3	10(3.2)	446	3.5	0	12.0
(EXT,30)	0.1	93	7.3	10(3.3)	381	4.1	1	14.4
	0.01	128	7.3	10(3.2)	446	3.5	0	12.0
(BDF,10)	0.1	85	7.3	10(5.1)	261	3.1	0	16.7
	0.01	132	7.3	10(5.3)	338	2.6	0	14.5
(BDF,30)	0.01	132	7.3	26(8.0)	305	2.3	0	19.4
PSODE	10^{-4}	94	5.9	1	616		0	6.5
	10^{-5}	127	7.4	1	829		0	6.5

Table 4. Results for the Van der Pol equation (11).

Method	Tol	N	nsd	$K_{\max}(K_{\text{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)	0.3	101	4.9	4(3.3)	410	2.9	49	14.4
	0.1	119	6.0	4(3.1)	432	2.5	40	12.3
	0.01	190	8.2	4(3.1)	514	2.0	43	9.4
	0.001	322	10.0	4(3.0)	673	1.9	23	7.3
(EXT,10)	0.3	105	5.1	10(6.3)	431	2.8	49	26.6
	0.1	126	6.3	10(5.6)	407	2.1	50	19.0
	0.01	194	8.1	10(6.0)	484	1.8	42	16.0
	0.001	324	10.0	10(6.7)	652	1.8	27	14.4
(EXT,30)	0.3	114	5.3	26(9.7)	425	2.2	60	36.9
	0.1	128	6.2	15(6.0)	415	2.1	53	20.5
	0.01	197	8.3	18(7.6)	476	1.6	48	19.3
	0.001	330	10.0	25(9.0)	650	1.6	38	18.7
PSODE	10^{-4}	132	6.3	1	883		26	6.7
	10^{-5}	184	7.4	1	1193		32	6.5
	10^{-7}	421	8.1	1	2670		39	6.3
	10^{-8}	626	8.7	1	3738		43	5.9

Table 5. Results for the Van der Pol equation (12).

Method	Tol	N	nsd	$K_{\max}(K_{\text{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)	0.1	191	6.3	4(2.6)	834	2.8	83	12.2
	0.01	288	7.9	4(2.3)	945	2.3	72	10.2
	0.001	471	9.8	4(2.6)	1264	2.4	36	7.9
(EXT,10)	0.1	181	6.5	10(4.5)	734	2.6	89	19.2
	0.01	289	7.7	10(4.8)	929	2.3	69	17.0
	0.001	477	9.7	10(5.3)	1260	2.3	48	15.2
(EXT,30)	0.1	190	6.5	14(4.7)	783	2.6	93	20.5
	0.01	294	7.8	19(6.0)	912	2.1	78	20.0
	0.001	479	9.7	23(6.9)	1214	2.1	44	18.3
PSODE	0.01	112	3.9	1	852		6	7.6
	10^{-4}	206	5.6	1	1430		36	6.9
	10^{-5}	281	6.9	1	1880		52	6.7
	10^{-6}	420	6.0	1	2739		59	6.5
	10^{-7}	693	7.8	1	4721		50	6.8
	10^{-8}	969	10.7	1	6310		42	6.5

Table 6. Results for the linear Prothero-Robertson problem (13).

Method	Tol	N	nsd	$K_{\max}(K_{\text{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)	0.01	40	9.5	4(2.0)	141	2.9	6	7.8
(EXT,10)	0.01	40	9.5	9(3.6)	141	2.9	6	13.7
(EXT,30)	0.01	40	9.5	9(3.6)	141	2.9	6	13.7
(BDF,10)	10^{-4}	69	8.8	10(7.5)	144	2.1	11	16.3
(BDF,30)	10^{-4}	69	8.8	30(15.8)	125	1.8	11	28.3
PSODE	10^{-6}	49	8.1	1	411		0	8.4
	10^{-8}	120	9.0	1	1066		0	8.9
	10^{-9}	176	10.2	1	1414		0	8.0

Table 7. Results for the electric inverter (14).

Method	Tol	N	nsd	$K_{\max}(K_{\text{av}})$	C_{eff}	j^*_{av}	N_{reject}	m_{av}
(EXT,4)	0.2	37	5.8	4(3.5)	169	3.5	13	17.0
	0.1	44	6.3	4(3.0)	171	3.0	12	13.8
	0.01	76	7.8	4(3.2)	206	2.2	17	9.7
	0.001	125	8.3	4(3.2)	272	1.9	14	7.9
	0.0001	217	9.6	4(3.1)	388	1.6	27	6.6
(EXT,10)	0.2	40	5.1	10(7.0)	160	2.5	22	28.6
	0.1	47	7.1	10(6.3)	158	2.4	12	22.2
	0.01	78	7.5	10(6.3)	186	2.0	16	16.0
	0.001	130	9.0	10(6.9)	276	1.9	16	15.6
	0.0001	223	9.7	10(7.0)	400	1.6	28	13.8
(EXT,30)	0.2	40	5.2	17(8.7)	163	2.5	20	35.0
	0.1	46	5.5	18(7.7)	154	2.5	12	26.5
	0.01	79	7.9	13(6.3)	196	1.9	16	16.5
	0.001	129	8.6	16(8.2)	269	1.4	24	18.4
	0.0001	224	9.8	20(9.5)	386	1.3	31	17.6
(BDF,10)	0.01	73	7.5	10(6.6)	172	2.4	16	16.6
	0.001	137	8.6	10(7.8)	270	2.0	15	16.4
	0.0001	287	10.3	10(8.8)	472	1.6	16	15.4
(BDF,30)	0.01	73	7.5	13(7.4)	169	2.3	16	18.2
	0.001	137	8.6	26(10.4)	268	2.0	15	21.2
	0.0001	287	10.3	25(13.8)	453	1.6	16	22.8
PSODE	10^{-4}	57	6.0	1	377		14	6.6
	10^{-6}	131	8.8	1	795		29	6.0
	10^{-7}	186	9.5	1	1089		32	5.8