# Parallelism in the Numerical Solution of Ordinary and Implicit Differential Equations

# Parallelism in the Numerical Solution of Ordinary and Implicit Differential Equations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam,
op gezag van de Rector Magnificus
Prof.dr. J.J.M. Franse
ten overstaan van een door
het college van dekanen ingestelde commissie
in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 21 mei 1997 te 15:00 uur

door

Wolter Adriaan van der Veen
geboren te Groningen

Parallelism in the Numerical Solution of Ordinary and Implicit Differential Equations

Promotor: Prof.dr. P.J. van der Houwen
Copromotor: Dr. B.P. Sommeijer

Faculteit: Wiskunde, Informatica, Natuurkunde en Sterrenkunde.

Het onderzoek dat tot dit proefschrift heeft geleid werd financieel ondersteund door de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (N.W.O) via de Stichting voor de Technische Wetenschappen (S.T.W.).
Printed at the Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

# Preface

This thesis deals with the numerical solution of ordinary and implicit differential equations. In several important areas of industrial research and design, such as circuit analysis and control engineering, one encounters equations of this type. Typically, these problems have a large dimension and require good stability properties of the integrator. This makes the integration quite expensive. To reduce the computational effort, a parallel approach was developed at CWI in 1991-1993. This approach was sufficiently promising to start a project "Parallel codes for circuit analysis and control engineering", funded by the Dutch Technology Foundation (STW). The Ph.D-students in this project are J.J.B. de Swart and the author of this thesis. This thesis consists of an introduction followed by the seven papers:

1. *Parallel Iteration Across the Steps of High-Order Runge-Kutta Methods for Nonstiff Initial Value Problems,*
   P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen,
   published in JCAM 60 (1995), 309-329.

2. *Parallelism Across the Steps in Iterated Runge–Kutta Methods for Stiff Initial Value Problems,*
   P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen,
   published in Numerical Algorithms 8 (1994), 293-312.

3. *Convergence Aspects of Step-Parallel Iteration of Runge-Kutta Methods,*
   W.A. van der Veen, J.J.B. de Swart and P.J. van der Houwen,
   published in JCAM 18 (1995), 397-411.

4. *Step-Parallel Algorithms for Stiff Initial Value Problems,*
   W.A. van der Veen,
   published in Computers & Mathematics with Applications 30(1995),11:9-23.

5. *Parallel Iterative Linear Solvers for Multistep Runge-Kutta methods,*
   E. Messina, J.J.B. de Swart and W.A. van der Veen,
   to be published in JCAM.

6. *The Solution of Implicit Differential Equations on Parallel Computers,*
   P.J. van der Houwen and W.A. van der Veen,
   submitted for publication.

7. *Waveform Relaxation Methods for Implicit Differential Equations,*
   P.J. van der Houwen and W.A. van der Veen,
   published in Advances in Computational Mathematics 1997.

Apart from these papers, the project resulted in:

- The CWI Test set for IVP solvers
- The code PSIDE (Parallel Software for Implicit Differential Equations)

Both were developed by W.M. Lioen, J.J.B. de Swart and the author of this thesis. The testset is available from internet via

http://www.cwi.nl/cwi/projects/IVPtestset.shtml.

For information on PSIDE we refer to the forthcoming thesis of J.J.B. de Swart entitled "Parallel Software for Implicit Differential Equations" or the user's guide and specification of PSIDE, which will appear as CWI report.

I want to thank all the people who contributed somehow to the realization of this thesis. First of all, I would like to express my gratitude to Prof.dr. P.J. van der Houwen en Dr. B.P. Sommeijer for their inspiring guidance. I want to thank Prof.dr. P.J. van der Houwen for his many contributions to the thesis and for all his help in the past four years. I want to thank Dr.B.P. Sommeijer for contributing to the thesis and for the many inspiring discussions.

Many thanks are expressed to J.J.B. de Swart for writing two papers together and for all his suggestions and advice in the past four years. I also want to thank him and W.M. Lioen for their cooperation in realizing the Testset and PSIDE

I am grateful to E. Messina from the university of Naples for writing a paper together, during her stay at CWI. Further, I want to thank Dr. W. Hoffmann for his interest in the subject and his constructive critiscim.

Finally, I want to thank CWI, STW and NWO for giving me the opportunity to prepare the thesis. I also want to thank the printshop of CWI for printing the thesis.

# Contents

# Chapter 1

# Introduction

## 1 Initial value problems for stiff equations

In industry, engineers often use simulations to study the behaviour of technical processes. For example, before actually manufacturing a computer chip, its behaviour is first simulated to see whether it functions properly. Many mathematical problems occurring in industrial simulations are partial differential equations, but in important areas, such as circuit analysis, chemical engineering and control systems design, engineers often encounter ordinary differential equations (ODE) and differential-algebraic equations (DAE). These equations are usually stiff, that is, they impose severe stability demands on the numerical integrator.

Well-known methods for integrating stiff equations are the backward differentiation formulas (BDF) and the implicit Runge-Kutta methods (IRK). Most software in use for these problems by engineers in industry is based on BDF methods. Although they are the cheapest methods for many problems, their stability properties are not that good. BDF methods combine high order with bad stability properties, which complicates the development of robust codes. On the other hand, implicit Runge-Kutta methods combine good stability properties with high order, which makes development of software more straightforward. In addition, the codes are more robust and can handle more difficult problems. In spite of this, they are rarely used in industry, because they are often much more expensive than BDFs, when implemented on sequential computers.

On parallel computers the situation is quite different. The BDF methods do not allow for problem-independent parallelism, while the IRKs possess intrinsically problem-independent parallelism. Therefore, it seems worthwhile to look for parallel methods based on IRK schemes, that inherit their good stability

properties and that allow for even more parallelism than already present. In this way, it turned out to be possible to develop parallel methods, that are on parallel computers more efficient than BDFs.
In the thesis of Sommeijer [12] (see also [4, 5]) the so-called PDIRK method was introduced. This method has the good stability properties of IRKs and achieves on the Alliant FX-4 a speed-up of about a factor 2 with respect to the well-known BDF code LSODE. This speed-up factor was the motivation for a research project entitled "*Parallel Codes for Circuit Analysis and Control Engineering* " which is funded by the Dutch Technology Foundation (STW). The PhD students in this group are J.J.B. de Swart and myself. An important spin-off of the research carried out by the group is the development of a test set consisting of real-life problems [9]. This test set is meant for testing ODE and DAE solvers and has already been used by many researchers in the field.
The research presented in this thesis concentrates on two main topics: (i) the introduction of parallelism across the steps in the PDIRK method and (ii) the improvement of PDIRK and its extension to the numerical integration of implicit differential equations and to waveform relaxation.
In the following, we give an outline of our investigations. In Section 2 and 3, we briefly review implicit Runge-Kutta methods and the solution of the nonlinear systems that are encountered in their application. In Section 4 we give the essentials of the PDIRK method. In Section 5 we review step-parallel methods that are based on PDIRK. As background for the second topic we discuss in Section 6 an improvement of PDIRK [3] and in Section 7 we consider multistep Runge-Kutta methods. The extension of PDIRK to implicit differential equations and waveform relaxation is reviewed in Section 8 and 9, respectively.

# 2 IRK methods

Consider the initial value problem

$$y'(t) = f(y(t)), \quad y(t_0) = y_0, \quad y \in R^d, \quad t_0 \leq t \leq t_{\text{end}}.$$

For obtaining an approximation $y_n$ to the solution at a step point $t_n$, the application of an implicit Runge-Kutta method consists of two phases. First, an approximation to the solution at a number of points in the interval $(t_{n-1}, t_n]$ is computed. The number of these points is called the number of stages and is denoted by $s$. The points themselves are given by $t_{n-1} + c_i h, \quad i = 1, \ldots s$, with $h = t_n - t_{n-1}$, and the approximations at these points are denoted by $y_{n,i}$. These stage values $y_{n,i}$, $i = 1, \ldots s$, are determined as the solution of the

$s$ equations of dimension $d$, which are of the form

$$y_{n,i} = y_{n-1} + h \sum_{j=1}^{s} a_{ij} f(y_{n,j}), \quad i = 1, \ldots, s.$$

Note, that these equations are coupled. Given the stage values, the approximation at the step point $t_n$ is given by

$$y_n = y_{n-1} + h \sum_{i=1}^{s} b_i f(y_{n,i}).$$

The matrix $A = (a_{ij})$, the vector $b = (b_i)$ and the vector $c = (c_i)$, all of dimension $s$, determine the Runge-Kutta method.
For a compact notation, we introduce the stage vector $Y = (y_{n,i})$, consisting of the $s$ stage values. In terms of this stage vector the Runge-Kutta scheme reads

$$\begin{aligned} Y &= e \otimes y_{n-1} + h(A \otimes I)F(Y), \\ y_n &= y_{n-1} + h(b^T \otimes I)F(Y). \end{aligned}$$

Here, $F(Y)$ is defined by $F(Y) := (f(Y_i))$, the matrix $I$ is the identity matrix of order $d$, $e$ is the vector of length $s$ that has all its components equal to one, and $\otimes$ denotes the Kronecker product defined by $(A \otimes B) = (a_{ij}B)$. In the following, $I$ denotes the identity matrix of either order $d$ or order $sd$.
For the integration of stiff problems, the method should have good stability properties, such as A-stability or L-stability [2]. Well-known IRK methods, that satisfy these demands are the L-stable Radau IIA methods of order $2s-1$ and the A-stable Gauss-Legendre methods of order $2s$. The Radau IIA method has the step point $t_n$ as one of its stage points. This is of special importance for differential-algebraic equations, because if all the algebraic equations are satisfied in all stage points, then they are automatically satisfied in the step point.
In the experiments reported in this thesis, we focus on the four-stage Radau IIA method. This has historical reasons. The PDIRK method was implemented on the Alliant FX-4, having four processors. To use all these processors effectively, the PDIRK method based on the four stage Radau IIA method was used.
In the following, we consider general IRK methods for which the $A$ matrix is full and has complex eigenvalues (like the Radau IIA methods and the Gauss-Legendre methods, for $s \geq 2$).

# 3 Solving the corrector equation

We have seen that, in the application of an IRK method in a step point, one has to solve the system

$$R(Y) := Y - e \otimes y_{n-1} - h(A \otimes I)F(Y) = 0.$$

Usually, this is accomplished by modified Newton iteration

$$(I - hA \otimes J)(Y^j - Y^{j-1}) = -R(Y^{j-1}),$$

where $J$ is the Jacobian matrix of $f$ evaluated at $(t_{n-1}, y_{n-1})$. Since $A$ is a full matrix, all the stage values are coupled. Assuming that direct solution methods are used, the LU-decomposition is the most expensive part, requiring $\frac{2}{3}s^3d^3$ operations. This can become quite expensive for values of $s$ that are used in practice ($3 \leq s \leq 6$).

There is a trick for making the LU-decomposition cheaper and in addition, it allows for parallelism. In the following we assume that $s$ is even. Instead of computing directly an LU-decomposition, the system is first transformed by a similarity transformation to a block-diagonal matrix having blocks of order $2d$, see [2]. Essentially, this similarity transformation puts the matrix $A$ into $2 \times 2$ block-diagonal form. Then the $s/2$ blocks, each of order $2d$, can be decomposed in parallel. Therefore, the costs of making an LU-decomposition on a *parallel* computer is only $\frac{16}{3}d^3$ operations, while the costs on a *sequential* computer is $\frac{8}{3}sd^3$. Here, we see that there is already a substantial amount of problem-independent parallelism in the IRK method.

There is a possibility for reducing the computational effort even further by using complex arithmetic. By employing the eigenvalue decomposition of the matrix $A$, the system is transformed to a complex block-diagonal system with blocks of size $d$. The LU-decomposition of this block-diagonal matrix is formed by decomposing the $s$ subsystems of order $d$. This can be done in parallel on $s$ processors, yielding a computational effort of $\frac{8}{3}d^3$ operations. Nevertheless, the computational effort is still much higher than the computational effort required for solving the systems of order $d$ that are encountered in the application of BDF methods. These are of the form

$$(I - h\gamma J)(Y^j - Y^{j-1}) = b,$$

with $b$ given and $\gamma$ a scalar. The LU-decomposition of this matrix requires only $\frac{2}{3}d^3$ operations. On the other hand, IRKs have much better stability properties. Therefore, it seems worthwhile to look for methods based on IRK schemes, which are more efficient on parallel computers than BDF methods by introducing additional parallelism.

# 4 PDIRK

If the matrix $A$ would have been a diagonal matrix, then the matrix $I - hA \otimes J$ occurring in the Newton iteration process is block-diagonal and we can solve the system by handling the $s$ diagonal blocks in parallel. However, since $A$ is a full matrix, the system under consideration is too expensive to treat directly. In order to reduce the computational effort we shall use relaxation. Suppose, we want to solve $Sx = b$, but an LU-decomposition of $S$ is too expensive. Hence, we split $S$ into $S = M + N$, and generate iterates given by $Mx^{k+1} + Nx^k = b$. In our case we split $I - hA \otimes J$ into $I - hD \otimes J$ and $h(D - A) \otimes J$, with $D$ a suitably chosen diagonal matrix. This leads to the iterative process given by

$$\begin{aligned} Y^{j,0} &= Y^{j-1}, \qquad (1) \\ (I - hD \otimes J)(Y^{j,\nu} - Y^{j-1}) &= h((A - D) \otimes J)(Y^{j,\nu-1} - Y^{j-1}) - R(Y^{j-1}), \\ Y^j &= Y^{j,r}. \end{aligned}$$

Here, $\nu$ runs from 1 to $r$. If this iteration process converges, then the iterate $Y^{j,\nu}$ converges to $Y^j$. This iteration scheme is called the *inner* iteration process and can be interpreted as an iterative linear system solver.
Depending on the quality of $D$, one iteration often already suffices, yielding the scheme

$$(I - hD \otimes J)(Y^j - Y^{j-1}) = -R(Y^{j-1}).$$

This last scheme can also be considered as a nonlinear system solver or as modified Newton, in which the matrix $A$ occurring in the Jacobian of $R(Y)$ has been replaced by the diagonal matrix $D$. It was introduced and analyzed in [4, 5] and was called the PDIRK method, which is the acronym for Parallel Diagonally Iterated Runge-Kutta method.
In each iteration, a system of the form $(I - hD \otimes J)x = b$ has to be solved. Since this system is block-diagonal, it decouples into $s$ systems of dimension $d$. The matrices in these subsystems are the same in each iteration and, moreover, can be decomposed in parallel. Another expensive operation is the evaluation of the right hand side $R(Y^{j-1})$. This amounts to performing the $f$-evaluations of the $s$ stage values and can again be done in parallel. Hence, for performing one PDIRK iteration the computational effort required consists of (i) the $s$ LU-decompositions, requiring $\frac{2}{3}d^3$ operations (only for the first iteration), (ii) $s$ function evaluations, effectively requiring only one function evaluation and (iii) one forward-backward substitution, effectively requiring $2d^2$ operations. In listing these costs, we assumed that $s$ processors are used.
Important for the performance of PDIRK is its convergence. If the process converges too slowly, then the method can still be quite expensive. In Section 6 we shall describe an improvement of the PDIRK scheme.

To study the convergence, we apply the PDIRK method to the test problem $y' = \lambda y$. We shall consider the convergence for stable problems only, that is, $h\lambda$ is in the left half plane. Define the iteration error by $\varepsilon^j = Y^j - Y$. One easily derives the error recursion $\varepsilon^j = Z(z)\varepsilon^{j-1}$, with $Z(z) = z(I - zD)^{-1}(A - D)$ and $z = h\lambda$. In [4, 5], $D$ was chosen such that the very stiff components in the iteration error are strongly damped. That is, $D$ was chosen such that $D$ minimizes $\rho(Z(-\infty)) = \rho(I - D^{-1}A)$. In [4] diagonal matrices with $\rho(I - D^{-1}A) \approx 0$ were given for the two, three and four stage Radau IIA method. In [8] matrices $D$ satisfying $\rho(I - D^{-1}A) = 0$ were determined for the $s$-stage Radau IIA methods with $2 \leq s \leq 8$. For these $D$ matrices it was shown that $\rho(Z(z)) \leq 1$ for $\mathrm{Re}(z) \leq 0$. Therefore, for all stable test problems the PDIRK methods converge.
Experiments with the four stage PDIRK method show a satisfactory convergence for nonlinear problems. The speed-up in comparison with LSODE has been measured for a large class of test problems on a four processor Alliant FX-4 shared-memory computer and was about a factor 2.

# 5 Step-parallel methods

In general, PDIRK requires relatively many iterations (the number roughly equals the order of the underlying IRK). This motivated us to apply the idea of step-parallelism introduced by Bellen et al. [1] to PDIRK (see Chapter 3). The usual way of solving ODEs is to iterate in a step point until a satisfactory approximation to the solution has been found. Only after finishing the iteration process in a step point, the iteration process in the next step point is started. In step-parallelism, one starts the iteration in the next step point, while the iteration in the preceding step point is still proceeding. That is, the iteration process in the next interval is already started as soon as the iterate in the preceding interval is sufficiently reliable. However, when the computations in this next step point are started, the iteration processes in all preceding intervals, that have not yet converged, still continue. In this way, iterations in several intervals are carried out simultaneously on the available processors. Of course, it may happen, that the iteration can be started in the next step point, but that there are no processors idle at the time. Hence, in step-parallelism a relatively large amount of parallelism can be exploited.
For step-parallel methods based on PDIRK, we prove in Chapter 3 and 4 a few theoretical results. Here, we state only the most general result. We apply the method to the test equation $y' = \lambda y$ and we make the following assumption on the step-parallel method. Whenever say $j^*$ iterations have been performed in a step point, we start the iteration process in the next step point, but we

also proceed with the iterations in the preceding step points that have not yet converged. In Chapter 4 we prove for this model situation the following convergence theorem.

**Theorem.** If PDIRK converges for the test problem, then the step-parallel version also converges for the test problem.

Hence, theoretically it converges, but if too many intervals are treated simultaneously, then the convergence speed may become too low. How severe this problem is, depends on the convergence properties of the PDIRK iteration process. Especially the initial convergence is important, which appears to be rather slow for PDIRK. Consequently, in the implementation of a step-parallel method based on PDIRK it is important that the iteration in the next step point is not started too early.
In an actual implementation one has to come up with a strategy that determines when it is save to start iterating in the next point using the current iterate in the present point (that might be of doubtful quality). This is the subject of Chapter 5. It was possible to develop a robust strategy, that gives speed-up with respect to PDIRK of a factor two. The number of intervals that can be treated simultaneously in an effective way is limited to four.
Until now we considered only stiff ODEs. In Chapter 2, we apply the idea of step-parallelism to the integration of nonstiff ODEs. Experiments show that the method gives a speed-up with respect to DOPRI8, that varies from three to eight. The number of intervals that can be treated simultaneously in an effective way, is limited to ten.

# 6 Improving PDIRK

The PDIRK method performs quite well in practice and its *asymptotic* convergence properties are rather good, but its *initial* convergence behaviour might be less satisfactory. More specific, the minimal number of iterations for which the PDIRK method becomes L-stable is relatively large (particular for large values of $s$). This motivated the search for more sophisticated methods which is reported in the thesis of De Swart (see also [3]).
In the PDIRK method, the matrix $I-hA\otimes J$ is split into the block-diagonal matrix $I-hD\otimes J$ and the matrix $h(D-A)\otimes J$. In [3] lower triangular matrices are considered instead of diagonal matrices $D$. Thus, the matrix $I-hA\otimes J$ is split into the lower block-triangular matrix $I-hT\otimes J$ and the matrix $h(T-A)\otimes J$. This splitting yields a relaxation method that is similar to (1). Performing only

one iteration of this linear system solver, yields the PTIRK method

$$(I - hT \otimes J)(Y^j - Y^{j-1}) = -R(Y^{j-1}).$$

If the lower triangular matrix $T$ has distinct eigenvalues then it is similar to a diagonal matrix and the lower block-triangular matrix $I - hT \otimes J$ can be transformed to a block-diagonal matrix. Since similarity transformations are cheap on parallel computers, the costs of the PTIRK method are comparable to the costs of the PDIRK method.
The triangular matrix $T$ is chosen such that the stiff components in the iteration errors are strongly damped and are removed after $s$ iterations. In addition, the eigenvalues must be distinct. In the thesis of De Swart, it is proved that for every collocation method such a matrix $T$ does exist.
The PTIRK method with four stages has been tested on a large number of test problems. It proves to be a more robust and a more stable method than PDIRK. Furthermore, the PDIRK approach does not work for six and eight stages, whereas PTIRK does work. Using PTIRK in step-parallel mode made strategy issues less critical and gives for several problems a speed-up relative to PDIRK across the steps of about 30 %, but for other problems there is no speed-up. However, for waveform relaxation methods discussed in Section 9, PTIRK works well, whereas PDIRK does not work at all.

# 7 Multistep Runge-Kutta methods

The basic idea of multistep RK methods is to make IRKs cheaper by adding a few back values. In this way, the high order can be maintained, while the number of stages is reduced and the method becomes less expensive. Such methods are called multistep Runge-Kutta methods. For a nice introduction to these methods we refer to [2] and the thesis of Schneider [11].
A well-known class of multistep Runge-Kutta methods that are suitable for the integration of stiff ODES are the multistep Radau methods. In applying these methods we again encounter systems of the form $I - hA \otimes J$ and we can use once more PTIRK-like methods. Two such methods are discussed in Chapter 6. The speed-ups of these methods are still to be determined; this is subject of future research.

# 8 Parallel methods for implicit differential equations

Next, we consider the initial value problem for implicit differential equations:

$$\phi(y', y) = 0, \quad y(t_0) = y_0, \quad y'(t_0) = y'_0.$$

It will be assumed that this equation has a unique solution $y(t)$. Furthermore, the Jacobian $\frac{\partial \phi}{\partial y'}$ may be singular. One way of applying Runge-Kutta methods, is to consider the formulation

$$\begin{aligned} y' &= z, \\ 0 &= \phi(z, y), \end{aligned}$$

which is analytically equivalent. Applying a Runge-Kutta method is now straightforward and yields

$$\begin{aligned} Y &= e \otimes y_{n-1} + h(A \otimes I)Z, \\ 0 &= \Phi(Z, Y). \end{aligned}$$

Here, the function $\Phi$ is defined by $\Phi(Z, Y) = (\phi(Z_i, Y_i))$ and $Z$ is the stage vector that approximates the derivatives in the stage points. This system of dimension $2sd$ can be reduced to the $sd$-dimensional system

$$0 = \Phi(h^{-1}(A^{-1} \otimes I)(Y - e \otimes y_{n-1}), Y),$$

or equivalently, to

$$R(Y) := h(A \otimes I)\Phi(h^{-1}(A^{-1} \otimes I)(Y - e \otimes y_{n-1}), Y) = 0.$$

In every step point, we have to solve the nonlinear system $R(Y) = 0$. Applying the modified Newton method gives

$$(I \otimes K + hA \otimes J)(Y^j - Y^{j-1}) = -R(Y^{j-1}), \tag{2}$$

where $K = \frac{\partial \phi}{\partial y'}$ and $J = \frac{\partial \phi}{\partial y}$. This modified Newton process has a similar form as the one in the ODE case.

In order to solve the linear system, we first transform it into either a suitable block-diagonal matrix or a lower block-triangular matrix with blocks of dimension $2d$. In the first case we solve the system in a block-Jacobi way and solve each of the subsystems by PTIRK-like methods. For details we refer to the thesis of De Swart and for its extension to IDEs see Chapter 7. In the case of a lower block-triangular matrix, the system is solved in a block Gauss-Seidel way. As in the block-Jacobi case the subsystems are solved by PTIRK-like methods.

This Gauss-Seidel method is developed and extended to IDEs in Chapter 7. The iteration process used to solve the subsystems is again called the *inner* iteration process.
Experiments have shown that, in the case of IDEs, only one inner iteration is often insufficient and that it is appropriate to determine the number of inner iterations dynamically. Furthermore, it turned out that the Jacobi method is the most efficient method. Based on this method we developed a code for the integration of implicit differential equations on shared-memory computers, called PSIDE, see [10]. For testing strategies and for testing the final performance the test set [9] was used. The speed-up of PSIDE is still to be determined and we refer to the forthcoming thesis of De Swart.

# 9 Waveform relaxation

As we have seen, the application of modified Newton gives rise to linear systems. In IDEs occurring in circuit analysis the system matrices are often close to lower block-triangular matrices or block-diagonal matrices, which allow for parallel solution methods. For such structured IDEs the waveform relaxation approach has been introduced by Lelarasmee et al. [6, 7].
The idea of waveform relaxation is as follows. Instead of solving the original IDE, we solve a sequence of IDEs whose numerical integration leads to systems that have the aforementioned structure. This is achieved by using relaxation. The original IDE is split into two parts. One part will be close to the original IDE but is required to lead to cheap linear systems in its numerical integration. Therefore, it is not expensive to treat this part in an implicit way. The other part will be treated explicitly, because it is rather small.
In waveform relaxation, we first substitute a predictive waveform in the explicit part, resulting in an IDE, that can be integrated much cheaper. Here a waveform is a function, that approximates the solution on a subinterval. We solve this IDE and we get a new waveform. This waveform is probably better than the predictive waveform, that we have put into the explicit part. Therefore, we repeatedly substitute this waveform in the explicit part to obtain a new IDE. This can be done as many times as is necessary.
Although the solution of the linear systems is considerably cheaper than the solution of the original system, they still contains the $A$ matrix, so that we can make the solution method even cheaper by applying the PTIRK approach. However, this introduces another iteration process in the waveform relaxation method and the question arises how severely the convergence of waveform relaxation is affected by the approximate solution of these linear systems.
In waveform relaxation, the idea of step-parallelism is also applicable. Wave-

form relaxation naturally allows the use of step-parallelism, without any algorithmic change. In this way, the method uses three types of parallelism: across the problem, across the steps and across the stage values.
As shown in the last Chapter (using several test problems from the literature), convergence is only mildly affected, when using the PTIRK method, but becomes quite dramatic for PDIRK methods. In fact so bad, that we refrained from using the PDIRK method in waveform relaxation.

## References

[1] A. Bellen. *Parallelism across the steps for difference and differential equations*, pages 22–35. Lecture Notes in Mathematics 1386. Springer-Verlag, 1987.

[2] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Springer-Verlag, 1996.

[3] P. J. van der Houwen and J. J. B. de Swart. Triangularly implicit iteration methods for ODE-IVP solvers. Technical Report NM-R9510, CWI, Amsterdam, 1995. To appear in: SIAM Journal on Scientific Computing, 18(1), January 1997.

[4] 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.

[5] P. J. van der Houwen and B. P. Sommeijer. Analysis of parallel diagonally implicit iteration of Runge–Kutta methods. *APNUM*, 11:169–188, 1993.

[6] E. Lelarasmee. *The waveform relaxation method for the time domain analysis of large scale nonlinear dynamical systems.* PhD thesis, Univ. of California, Berkeley, 1982.

[7] E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli. The waveform relaxation method for time domain analysis of large scale integrated circuits. *IEEE Trans. CAD IC Syst.*, 1:131–145, 1982.

[8] W. M. Lioen. On the diagonal approximation of full matrices. Technical Report NM-R9518, CWI, Amsterdam, 1995. To appear in: JCAM.

[9] W. M. Lioen, J. J. B. de Swart, and W. A. van der Veen. Test set for IVP solvers. Report NM-R9615, CWI, Amsterdam, 1996. WWW version available at URL http://www.cwi.nl/cwi/projects/IVPtestset.shtml.

[10] W. M. Lioen, J. J. B. de Swart, and W. A. van der Veen. Specification of PSIDE. Report, CWI, Amsterdam, 1997. In preparation.

[11] S. Schneider. *Intégration de systèmes d'équations différentielles raides et différentielles-algébriques par des méthodes de collocations et méthodes générales linéaires.* PhD thesis, Université de Genéve, 1994.

[12] B. P. Sommeijer. *Parallelism in the numerical integration of initial value problems.* PhD thesis, University of Amsterdam, 1992.

# Chapter 2

## 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)

$$\boldsymbol{y}'(t) = \boldsymbol{f}(\boldsymbol{y}(t)), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y}, \boldsymbol{f} \in \mathbb{R}^d \tag{1.1}$$

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

* Corresponding author.

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]):

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

Here $h$ denotes the stepsize, the matrix $B$ contains the RK parameters, and $\boldsymbol{F}(\boldsymbol{Y}_n)$ contains the derivative values $(\boldsymbol{f}(\boldsymbol{Y}_{n,i}))$, where $\boldsymbol{Y}_{n,i}$ denote the $d$-dimensional components of the (extended) stage vector $\boldsymbol{Y}_n$ (because of the GLM representation (2.1a), the RK solution at the step points is lumped into $\boldsymbol{Y}_n$). It will be assumed that (2.1a) possesses $s$ implicit stages and one explicit stage. The component of $\boldsymbol{Y}_n$ corresponding to the explicit stage approximates the exact solution at the step point $t_n = t_{n-1} + h$. The other stage vector components $\boldsymbol{Y}_{n,i}$ represent numerical approximations at the intermediate points $t_{n-1} + c_i h$, where $\boldsymbol{c} = (c_i) = B\boldsymbol{e}$, $\boldsymbol{e}$ 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

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

where $A$ and $\boldsymbol{b}$ 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 $\boldsymbol{Y}_0 = \boldsymbol{e} \otimes \boldsymbol{y}_0$. In the following, the dimension of $I$ and $\boldsymbol{e}$ 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** $m$
    $\boldsymbol{Y}_n^{(j)} = (E \otimes I)\boldsymbol{Y}_{n-1}^{(m)} + h(B \otimes I)\boldsymbol{F}(\boldsymbol{Y}_n^{(j-1)})$, (2.2)

where $m$ is the number of iterations, $\boldsymbol{Y}_0^{(m)} = \boldsymbol{e} \otimes \boldsymbol{y}_0$, and $\boldsymbol{Y}_n^{(0)}$ is to be provided by a predictor formula. Evidently, if (2.2) converges, then it converges to the corrector solution $\boldsymbol{Y}_n$.

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 $\boldsymbol{Y}_n^{(j)}$ 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 $\boldsymbol{F}(\boldsymbol{Y}_n^{(j-1)})$, 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 $\boldsymbol{f}$ 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 $\boldsymbol{G}(\boldsymbol{Y}) = \mathbf{0}$, where $\boldsymbol{Y}$ represents the vector containing *all numerical approximations in the whole integration interval*, and solves this system for $\boldsymbol{Y}$ 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), $\boldsymbol{Y}$ represents the $N$ stage vectors $\boldsymbol{Y}_n$, $n = 1, \dots, N$.

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

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

Here, we have $\boldsymbol{Y}_0^{(j)} = \boldsymbol{e} \otimes \boldsymbol{y}_0$ 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 $\boldsymbol{Y}_n^{(j)}$ to be equal for all iterates. Therefore, here and in the following, the costs of computing the prediction $\boldsymbol{Y}_n^{(0)}$ are assumed to be negligible. Thus, given the initial guesses $\boldsymbol{Y}_n^{(0)}$, $n = 1, \ldots, N$, first all stage vectors $\boldsymbol{Y}_n^{(1)}$ are computed concurrently, then all $\boldsymbol{Y}_n^{(2)}$, 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 $\boldsymbol{Y}_n^{(1)}$, a situation that can easily occur in the case of large integration intervals. This is caused by the fact that the prediction $\boldsymbol{Y}_n^{(0)}$ is either based on mere extrapolation of the initial value $\boldsymbol{y}_0$ 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 $\boldsymbol{Y}_n^{(1)}$.

### *2.1. The PIRKAS GS method*

Consider the recursion

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

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 $\boldsymbol{Y}_n^{(j)}$ 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

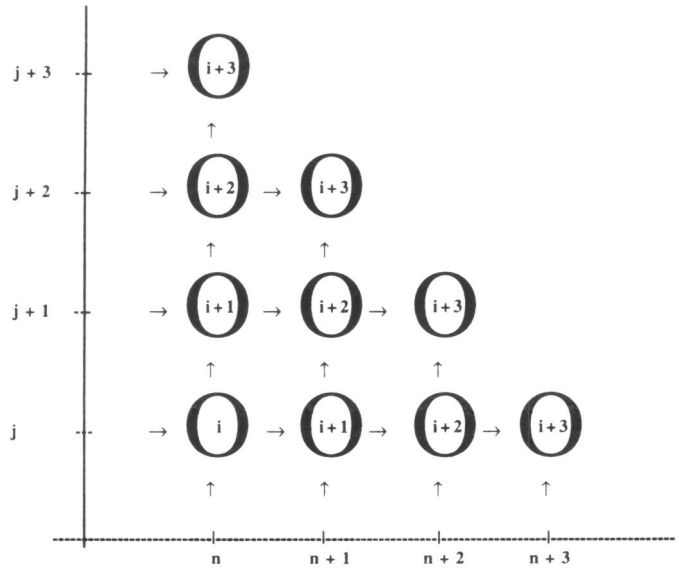


Fig. 1. Grid of $Y_n^{(j)}$ iterates in the $(n, j)$-plane.

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 $\boldsymbol{Y}_n^{(i-n)} = \boldsymbol{Y}_n^{(0)}$ by means of the predictor formula (see Section 2.2)
  **ELSE** **IF** $n = 0$ **THEN** $\boldsymbol{Y}_n^{(i-n)} = \boldsymbol{e} \otimes \mathbf{y}_0$ (2.6b)
    **ELSE** Compute $\boldsymbol{Y}_n^{(i-n)}$ 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 $N_{\text{seq}} = N + m$ 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 $\boldsymbol{Y}_n^{(1)}$, $n = 1, \ldots, N$ are computed, next $\boldsymbol{Y}_n^{(2)}$, $n = 1, \ldots, N$, 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 $\lambda$ is assumed to run through the spectrum of $\partial \boldsymbol{f}/\partial \boldsymbol{y}$. 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 $\boldsymbol{Y}_n^{(1)}$, $n = 1, 2, \ldots, N$. 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 $\boldsymbol{Y}_n^{(1)}$ and the region of convergence of the correction formula (2.4).

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

| $i$ | $t_1$ | $t_2$ | $t_3$ | $\ldots$ | $t_m$ | $t_{m+1}$ | $\ldots$ | $t_{N-m}$ | $t_{N-m+1}$ | $\ldots$ | $t_N$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | $\boldsymbol{Y}_1^{(1)}$ | | | | | | | | | | |
| 3 | $\boldsymbol{Y}_1^{(2)}$ | $\boldsymbol{Y}_2^{(1)}$ | | | | | | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | | | | |
| $m+1$ | $\boldsymbol{Y}_1^{(m)}$ | $\boldsymbol{Y}_2^{(m-1)}$ | $\boldsymbol{Y}_3^{(m-2)}$ | $\vdots$ | $\boldsymbol{Y}_m^{(1)}$ | | | | | | |
| $m+2$ | | $\boldsymbol{Y}_2^{(m)}$ | $\boldsymbol{Y}_3^{(m-1)}$ | $\vdots$ | $\boldsymbol{Y}_m^{(2)}$ | $\boldsymbol{Y}_{m+1}^{(1)}$ | | | | | |
| $\vdots$ | | | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | |
| $N+1$ | | | | | | | | | $\boldsymbol{Y}_{N-m+1}^{(m)}$ | $\vdots$ | $\boldsymbol{Y}_N^{(1)}$ |
| $\vdots$ | | | | | | | | | | | $\vdots$ |
| $N+m$ | | | | | | | | | | | $\boldsymbol{Y}_N^{(m)}$ |

*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, $\boldsymbol{Y}_n^{(1)}$ is computed by means of information coming from the iterate $\boldsymbol{Y}_{n-1}^{(1)}$. As already observed, we want all iterations of comparable sequential computational complexity, so that we are led to the predictor formula $\boldsymbol{Y}_n^{(0)} = (E_n^* \otimes I)\boldsymbol{Y}_{n-1}^{(1)}$, to obtain

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

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 $\boldsymbol{Y}_n^{(1)}$, 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 $\boldsymbol{Y}_{n,i}$ 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 $\boldsymbol{Y}_{n-1,i}^{(1)}$ yields predictions $\boldsymbol{Y}_{n,i}^{(0)}$ 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

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

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

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

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

$$\boldsymbol{Y}_n^{(1)} = E\boldsymbol{Y}_{n-1}^{(1)} + zBE^*\boldsymbol{Y}_{n-1}^{(1)}, \quad z := \lambda h.$$

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

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

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 $C := \boldsymbol{e}\boldsymbol{b}^T$, 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* $\mathbb{C} := \{z: \rho(zA) < 1\}$.

Recalling that $\lambda$ 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 $\varkappa(N, j)/(|\lambda|T)$ 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 $\delta$ 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 $\kappa(\tilde{M}) := \|\tilde{M}\|\,\|\tilde{M}^{-1}\|$,

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

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 $\|\cdot\|_\varkappa$, the following result can be derived.

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

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

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 $\boldsymbol{Y}_n^{(1)}$. 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 $\boldsymbol{Y}_{n-1}^{(j)}$ has the required properties to serve as a starting point to move to the next step point for computing $\boldsymbol{Y}_n^{(1)}$. This can be achieved by using in (2.7) the predictor formula

$$\boldsymbol{Y}_n^{(0)} = (E_n^* \otimes I)\boldsymbol{Y}_{n-1}^{(j^*)}, \tag{3.1}$$

to obtain

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

where $j^*$ is such that $\boldsymbol{Y}_{n-1}^{(j^*)}$ 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 $\boldsymbol{Y}_n^{(j)}$, 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 $\boldsymbol{Y}_n^{(j)}$, $n = 1, \ldots, N$, simultaneously, we shall iterate on the last $P$ iterates $\boldsymbol{Y}_v^{(j)}$, $v = n - P, \ldots, n - 1$, that do not yet have the corrector accuracy. In fact, we only proceed to the next step point if $\boldsymbol{Y}_{n-P}^{(j)}$ has the corrector accuracy and if $\boldsymbol{Y}_{n-1}^{(j)}$ is a safe starting point for computing

$\boldsymbol{Y}_n^{(0)}$. 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

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

Thus, first, we require that at $t_{n-P}$ the corrector is approximately solved by the iterate $\boldsymbol{Y}_{n-P}^{(j)}$ 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 $\boldsymbol{Y}_v^{(j)}$, $v = n - P + 1, \ldots, n - 1$ 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 $N_{\mathrm{seq}}$ 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

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

where $\boldsymbol{Y}_n^{(j)} = \boldsymbol{Y}_n^{(m(t_n))}$ 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

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

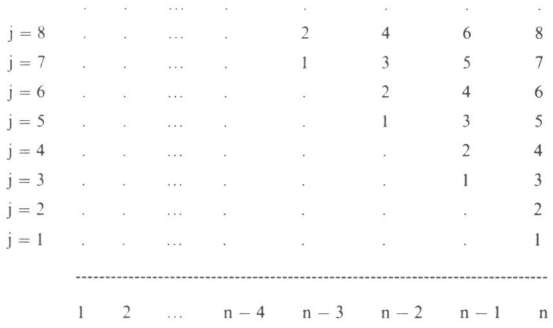


Fig. 2. Iteration index $i$ in the case $j^* = 2$.

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 $\tau_{n-1}$. 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

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

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 $\boldsymbol{Y}_{n-1}^{(0)}$ (obtained by extrapolation from $\boldsymbol{Y}_{n-2}^{(j^*)}$) and $\boldsymbol{Y}_{n-1}^{(1)}$, 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 $\Delta$, 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 $N_{\mathrm{seq}}$, 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 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $P$ | $\text{TOL}_{\text{pred}}$ | $\Delta$ | $N$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $m^*$ | $\Delta$ | $N$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $m^*$ |
| 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 $\text{TOL}_{\text{pred}}$. 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 $\text{TOL}_{\text{pred}}$ (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 $\text{TOL}_{\text{pred}} = 10^{-1}$. 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 $\text{TOL}_{\text{pred}} = 10^{-1}$

| $s$ | TOL | $\Delta$ | $N$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $m^*$ |
|---|---|---|---|---|---|---|
| 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 $N_{\text{seq}}$ 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* $\Delta$-values, the corresponding $N_{\text{seq}}$-values, obtained by interpolation. For the same $\Delta$-values, we calculate the values of $N_{\text{seq}}$ 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 $N_{\text{seq}}$).

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 $\text{TOL}_{\text{pred}}$. If $\text{TOL}_{\text{pred}}$ 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.

# Chapter 3

# Parallelism across the steps in iterated Runge–Kutta methods for stiff initial value problems*

P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen
*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

Communicated by J.C. Butcher
Received 26 November 1993; revised 3 September 1994

For the parallel integration of stiff 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-parallel approach received some attention in the case of Runge–Kutta based methods. For these methods, 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. 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 step-parallel iteration process proposed in the present paper is less massively parallel, but turns out to be sufficiently robust to solve the four-stage Radau IIA corrector used in our experiments within a few effective iterations per step and to achieve speed-up factors up to 10 with respect to the best sequential codes.

**Keywords:** Numerical analysis, Runge–Kutta methods, parallelism.

**Subject classification:** G.1.7.

## 1. Introduction

Recently, various attempts have been made to solve stiff initial value problems (IVPs)

$$\boldsymbol{y}'(t) = \boldsymbol{f}(\boldsymbol{y}(t)), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \quad \boldsymbol{y}, \boldsymbol{f} \in \mathbb{R}^d, \tag{1.1}$$

on parallel computers. Using the familiar terminology of parallelism "across the problem", "across the steps" and "across the method", we mention the *problem-parallel* methods based on wave form relaxation (cf. the survey paper of Burrage

* The research reported in this paper was partly supported by the Technology Foundation (STW) in the Netherlands.

[3]), the *step-parallel* methods of Bellen and coworkers [1,2] and Chartier [6], and the *method-parallel* solvers proposed in [10] based on parallel iteration of Runge–Kutta (RK) methods. To some extent, these three types of parallelism are orthogonal in the sense that they can often be combined. In this paper, we shall be concerned with step-parallelism.

Our starting point is a stiff IVP method that is both highly accurate and highly stable. This method is used as a corrector that is solved to convergence using parallel iteration techniques. In the selection of a suitable corrector, we are automatically led to the classical implicit Runge–Kutta methods such as the Radau IIA methods. These methods fulfil the requirements of accuracy and stability and belong to the best correctors for stiff problems. For the iteration method we choose the PDIRK (Parallel Diagonally Implicit RK) approach developed in [10] that solves the RK corrector by diagonally implicit iteration using $s$ processors, $s$ being the number of stages of the corrector. In [10], we advocated alternative correctors (called Lagrange correctors) which possessed stage order $s + 1$, whereas $s$-stage Radau methods have only stage order $s$. Since the stage order is important for the accuracy in many stiff problems and because the number of processors equals $s$, the Lagrange correctors may have advantages if the number of processors is small. However, recent developments indicate that the number of processors is no longer an important issue. Therefore, we adopt the Radau IIA methods as the correctors to be used in this paper. Using a predictor based on extrapolation of preceding stage values and the four-stage Radau IIA corrector, we obtained for the PDIRK approach a speed-up factor of about 2 with respect to the best sequential codes for stiff problems, viz. the variable order LSODE code and the fifth-order RADAU5 code [9]. An interesting feature of the PDIRK-based code (called PSODE in [16]) is the highly efficient performance of high-order correctors in the low accuracy range. Hence, assuming that sufficiently many processors are available, we may equally well use a Radau corrector with more than four stages without increasing the sequential costs, while the high order is effective both in the low and high accuracy range.

A drawback of the PDIRK methods is that the number of iterations needed to achieve corrector accuracy is still high (about the order of the corrector). To reduce the number of iterations, we introduced preconditioning into the PDIRK methods by which the number of iterations reduces substantially (cf. [11]). In this paper, we apply step-parallelism to the PDIRK methods. The analysis given here partly parallels the derivations in [13] for nonstiff problems.

## 2. Parallelism across the steps

Following [13], we write the RK method in the General Linear Method (GLM) form introduced by Butcher [4] (see also [5, p. 340]):

$$\boldsymbol{Y}_n = (E \otimes I_d)\, \boldsymbol{Y}_{n-1} + h_n (A \otimes I_d)\, \boldsymbol{F}(\boldsymbol{Y}_n), \quad n = 1, \ldots, N. \tag{2.1a}$$

Here, $h_n$ denotes the stepsize $t_n - t_{n-1}$, the matrix $A$ contains the RK parameters, and $\boldsymbol{F}(\boldsymbol{Y}_n)$ contains the derivative values $(\boldsymbol{f}(\boldsymbol{Y}_{n,i}))$, where $\boldsymbol{Y}_{n,i}$, $i = 1, 2, \ldots, s$, denote the $d$-dimensional components of the stage vector $\boldsymbol{Y}_n$. In this paper we will assume that (2.1a) possesses $s$ implicit stages and that the last stage corresponds to the step point $t_n$ (e.g. Radau IIA type methods). The first $s-1$ stage vector components $\boldsymbol{Y}_{n,i}$ represent numerical approximations at the intermediate points $t_{n-1} + c_i h_n$, $i = 1, 2, \ldots, s-1$, where $\boldsymbol{c} = (c_i) = A\boldsymbol{e}$, $\boldsymbol{e}$ being the vector with unit entries. In (2.1a), the matrix $E$ is of the form

$$E := \begin{pmatrix} 0 & \cdots & 0 & 1 \\ \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdots & \cdot & \cdot \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \tag{2.1b}$$

the matrix $I_d$ is the $d$-by-$d$ identity matrix, $\otimes$ denotes the Kronecker product, and we define $\boldsymbol{Y}_0 = \boldsymbol{e} \otimes \boldsymbol{y}_0$. In the following, the dimension of $I$ and $\boldsymbol{e}$ may change, but will always be clear from the context.

We approximate the solution $\boldsymbol{Y}_n$ of (2.1) by successive iterates $\boldsymbol{Y}_n^{(j)}$ satisfying the iteration scheme

$\boldsymbol{Y}_n^{(1)}$ defined by a predictor formula,

$$\boldsymbol{Y}_n^{(j)} - h_n(D \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(j)}) = (E \otimes I_d)\,\boldsymbol{Y}_{n-1}^{(q(n-1,j))} + h_n((A - D) \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(j-1)}),$$
$$j = 2, \ldots, m(t_n), \tag{2.2}$$

where $n = 1, 2, \ldots, N$ and $\boldsymbol{Y}_0^{(j)} = \boldsymbol{e} \otimes \boldsymbol{y}_0$ for all $j$. The predictor formula and the integer-valued function $q(n, j)$ will be discussed below. The number of iterations $m(t_n)$ performed at $t_n$ is defined by the condition that for $j = m(t_n)$ the iterates $\boldsymbol{Y}_n^{(j)}$ *numerically* satisfy the corrector equation (2.1) (evidently, if the iterates $\boldsymbol{Y}_n^{(j)}$ satisfying (2.2) converge to fixed vectors $\boldsymbol{V}_n$ as $j \to \infty$, then $\boldsymbol{V}_n = \boldsymbol{Y}_n$). The matrix $D$ will be assumed to be diagonal with $s$ positive diagonal entries. In the case of the $s$-stage Radau IIA correctors ($s = 2, 3, 4$), suitable matrices $D = D_s$ have been derived in [10]. For future reference, these matrices are here reproduced:

$$D_2 = \frac{1}{30}\begin{pmatrix} 20 - 5\sqrt{6} & 0 \\ 0 & 12 + 3\sqrt{6} \end{pmatrix},$$

$$D_3 = \begin{pmatrix} \dfrac{4365}{13624} & 0 & 0 \\ 0 & \dfrac{1032}{7373} & 0 \\ 0 & 0 & \dfrac{1887}{5077} \end{pmatrix}, \tag{2.3}$$

$$D_4 = \begin{pmatrix} \frac{3055}{9532} & 0 & 0 & 0 \\ 0 & \frac{531}{5956} & 0 & 0 \\ 0 & 0 & \frac{1471}{8094} & 0 \\ 0 & 0 & 0 & \frac{1848}{7919} \end{pmatrix}.$$

Irrespective of the definition of the function $q(n, j)$, the correction formula (2.2) possesses *parallelism across the method*, because the diagonal structure of the matrix $D$ enables us to compute the components of $\boldsymbol{Y}_n^{(j)}$ in parallel. In addition, a suitable definition of the function $q(n, j)$ may determine an ordering by which the iterates $\boldsymbol{Y}_n^{(j)}$ are computed that facilitates *parallelism across the steps*. We shall discuss various options.

2.1. *Order of computation of the iterates*

The conventional PC approach is defined by

$$q(n, j) = m(t_n),$$

for all $n$ and $j$. By this definition, the only possibility is to compute first the iterates $\boldsymbol{Y}_1^{(j)}, j = 1, 2, \ldots, m(t_1)$, next the iterates $\boldsymbol{Y}_2^{(j)}, j = 1, 2, \ldots, m(t_2)$, etc. This ordering generates the PDIRK method of [12] and [10]. Thus, representing the iterates by points in the $(n, j)$-plane, the PDIRK method computes the iterates *column-wise*. Obviously, this method does not allow for parallelism across the steps. If the predictor formula defining $\boldsymbol{Y}_n^{(1)}$ requires the same sequential costs as the correction formula in (2.2), then the sequential computational complexity of the PDIRK method is given by $N_{\text{seq}} = \Sigma_n m(t_n)$, $N_{\text{seq}}$ denoting the number of implicit systems to be solved.

A second option defines

$$q(n, j) = j - 1, \quad j > 1. \tag{2.4}$$

In this case, there are various possibilities in the ordering by which the iterates can be computed, leading to the same set of iterates. For example, it can again be done column-wise, but also *row-wise*. In the latter case, the iteration scheme {(2.2), (2.4)} may be considered as Jacobi-type iteration possessing a large degree of parallelism across the steps, because for fixed $j$, all iterates $\boldsymbol{Y}_n^{(j)}$, $n = 1, 2, \ldots, N$, can be computed concurrently. Therefore, it will be called the PDIRKAS J method (PDIRK Across the Steps using Jacobi iteration). The sequential computational complexity of the PDIRKAS J method is reduced to the sequential costs of computing all initial iterates $\boldsymbol{Y}_n^{(1)}$ and the sequential costs of solving $\max_n\{m(t_n)\} - 1$ implicit systems. In actual application, one wants to limit the sequential costs of the predictor formula. In the extreme case, one sets $\boldsymbol{Y}_n^{(1)} = \boldsymbol{e} \otimes \boldsymbol{y}_0$ for all $n$, so that $N_{\text{seq}} =$

$\max_n\{m(t_n)\} - 1$. The PDIRKAS J method using this strategy has similarities with the step-parallel methods studied by Bellen and co-workers [1,2]. However, such a PDIRKAS J method is expected to exhibit poor convergence due to the inaccuracy of $\boldsymbol{Y}_n^{(1)}$ as $n$ increases and can only be applied on small subintervals (windows). A more robust approach computes $\boldsymbol{Y}_n^{(1)}$ by a predictor formula of at least order one that is sufficiently stable (see section 2.2). Assuming that this predictor formula requires the same sequential costs as the correction formula in (2.2), the total sequential computational costs are given by $N_{\text{seq}} = N - 1 + \max_n\{m(t_n)\}$. Initially, the number of processors needed in this strategy is $sN$. However, in an actual implementation, iteration at a particular point $t_n$ will be stopped as soon as the corrector solution is obtained within some given tolerance (see section 4), so that the number of processors needed will gradually decrease.

Convergence will often be improved substantially by defining

$$q(n, j) = j. \tag{2.5}$$

Again, many algebraic equivalent orderings are possible (i.e., orderings that generate the same set of iterates). But now, neither the column-wise, nor the row-wise ordering does allow for step-parallelism. The only ordering by which a certain amount of step-parallelism is achieved, computes the iterates along the diagonals $n + j =$ constant, that is, all iterates $\boldsymbol{Y}_n^{(j)}$ with $n + j$ constant are computed concurrently. If we again restrict our considerations to predictor formulas that are equally expensive as the correction formula and if we assume that the iteration process at the end point of the integration interval is stopped only if iteration at all preceding step points has converged, then the total sequential computational costs are now given by $N_{\text{seq}} = N - 1 + m(t_N)$, where again the number of processors is at most $sN$ (but usually less than $sN$ as we saw in our previous discussion of the Jacobi iteration strategy). The iteration scheme defined by (2.5) may be considered as Gauss–Seidel-type iteration and the corresponding integration method will therefore be called the PDIRKAS GS method. We remark that the PDIRKAS GS method {(2.2), (2.5)} is the stiff version of the PIRKAS GS method developed in [13].

In the remainder of this paper, we analyse the PDIRKAS J and PDIRKAS GS methods.

### 2.2. *The predictor formula*

There are several possibilities in defining a predictor formula for the PDIRKAS method. An implementationally convenient choice defines $\boldsymbol{Y}_n^{(1)}$ by applying a backward differentiation formula (BDF) to the preceding iterate $\boldsymbol{Y}_{n-1}^{(1)}$ to obtain the implicit *stage vector* predictor formula

$$\boldsymbol{Y}_n^{(1)} - h_n(D^* \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(1)}) = (E^* \otimes I_d)\boldsymbol{Y}_{n-1}^{(1)}, \tag{2.6}$$

where $D^*$ is assumed to be diagonal. If we choose $D^* = D$, then we can achieve predictor order $q = s - 1$, while the predictor formula and the correction formula

share the same LU decomposition. If both $D^*$ and $E^*$ are defined by order conditions, then we have order $q = s$. However, we need an additional set of $s$ processors in order to compute the LU-decompositions for the predictor formula concurrently with those for the correction formula.

An alternative to (2.6) defines $\boldsymbol{Y}_n^{(1)}$ by applying a BDF to preceding step values $(E \otimes I_d)\boldsymbol{Y}_{n-1}^{(1)}$, $(E \otimes I_d)\boldsymbol{Y}_{n-2}^{(1)}, \ldots$. For example, we may define the *step point* predictor formula

$$\boldsymbol{Y}_n^{(1)} - h_n(D^* \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(1)}) = (E_1 \otimes I_d)\boldsymbol{Y}_{n-1}^{(1)} + (E_2 \otimes I_d)\boldsymbol{Y}_{n-2}^{(1)}, \tag{2.7}$$

where $D^*$ is again diagonal, and where the first $s-1$ columns of $E_1$ and $E_2$ vanish. Locally, this formula is at most third-order accurate. If $D^* = D$, then only second-order local accuracy can be achieved.

## 3. Stability and convergence

The stability region and the convergence region of the PDIRKAS methods will be discussed for the familiar basic test equation $y'(t) = \lambda y(t)$, where $\lambda$ is assumed to run through the spectrum of $\partial f/\partial y$. With respect to this test equation, the stability properties of the PDIRKAS method are determined by the stability of the predictor–corrector pair and the convergence properties of the iteration process. Unlike the situation in conventional PC methods, step-parallel methods as considered here, require the predictor to be stable for integration over the whole interval (row-wise ordering of the iterates). Assuming that the underlying corrector is unconditionally stable (with respect to the basic test equation), the stability region of the PDIRKAS method is the intersection of the region of convergence of the correction formula and the stability region of the predictor formula. At first sight, the stage value predictor formula (2.6) is more attractive, because of its higher order (we recall that the predictor order $q$ equals $s$ or $s-1$), whereas the step point predictor formula (2.7) is at most second-order accurate. However, (2.6) is less stable than (2.7). To see this, we consider the case where all coefficients are determined by order conditions. Furthermore, let the stepsizes be constant, i.e. $h_n = h$. Then, each of the $s$ components of $\boldsymbol{Y}_n^{(1)}$ defined by (2.6) may be considered as the result of applying an $s$-step BDF with the $s+1$ abscissas $\{t_{n-1} + c_i h, i = 1, \ldots, s; t_{n-1} + h + c_k h\}$ where $k = 1, \ldots, s$. In the case (2.7), each component of $\boldsymbol{Y}_n^{(1)}$ is defined by a two-step BDF with abscissas $\{t_{n-2}, t_{n-2} + h, t_{n-2} + h + c_k h\}$ where $k = 1, \ldots, s$. BDFs with non-uniformly distributed abscissas have been investigated in [8] and were shown to lead to poor stability regions if the spacing of the abscissas is *increasing*. Since in general the spacing of the last two abscissas in formula (2.6) is relatively large for $k > 1$, we cannot expect that (2.6) is sufficiently stable, whereas (2.7) is expected to be L-stable, because its stepsizes are nonincreasing. We also considered the case of (2.6) with $D^* = D$ and we did prove the existence of a family of first-order

predictors which are L($\alpha$)-stable for the two-, three- and four-stage Radau IIA correctors using the matrices $D$ as given in (2.3). For example, for the four-stage Radau IIA corrector we computed the angle $\alpha$ and found $\alpha \approx 70°$. Because the stability of the predictor formula is crucial in step-parallel methods, we decided to use the second-order, L-stable step point predictor formula (2.7) with $D^*$, $E_1$ and $E_2$ defined by order conditions.

### 3.1. *Region of convergence of the correction formula*

In this section, we shall derive the region of convergence for the recursion (2.2) when applied to the test equation. Let us define the stage vector iteration errors

$$\epsilon_n^{(j)} := \boldsymbol{Y}_n^{(j)} - \boldsymbol{Y}_n. \tag{3.1}$$

Subtracting (2.1) and (2.2), we find the linear recursion

$$\begin{aligned} \epsilon_n^{(j)} &= K_n \epsilon_{n-1}^{(q(n-1,j))} + Z_n \epsilon_n^{(j-1)}, \\ K_n &:= (I - z_n D)^{-1} E, \\ Z_n &:= z_n D (I - z_n D)^{-1} (D^{-1} A - I), \\ z_n &:= \lambda h_n, \end{aligned} \tag{3.2}$$

where $n = 1, \ldots, N$. We shall study the convergence of the iteration error vectors

$$\epsilon^{(j)} := \begin{pmatrix} \epsilon_1^{(j)} \\ \epsilon_2^{(j)} \\ \cdots \\ \epsilon_n^{(j)} \end{pmatrix}, \quad z := (z_1, \ldots, z_n)^{\mathrm{T}}. \tag{3.3a}$$

In particular, we are interested in the rate of convergence of the error vectors as function of $n$. The recursion (3.2) can be represented in the form

$$\epsilon^{(j)} = Q(z)\epsilon^{(j-1)} = Q(z)^{j-1}\epsilon^{(1)}, \tag{3.3b}$$

where in the case of the PDIRKAS J and PDIRKAS GS methods the $n$-by-$n$ block iteration matrix $Q(z)$ is respectively given by

$$Q_{\mathrm{J}}(z) := \begin{pmatrix} Z_1 & O & O & O & \cdots \\ K_2 & Z_2 & O & O & \cdots \\ O & K_3 & Z_3 & O & \cdots \\ O & O & K_4 & Z_4 & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix},$$

$$Q_{GS}(z) := \begin{pmatrix} Z_1 & O & O & O & \cdots \\ K_2Z_1 & Z_2 & O & O & \cdots \\ K_3K_2Z_1 & K_3Z_2 & Z_3 & O & \cdots \\ K_4K_3K_2Z_1 & K_4K_3Z_2 & K_4Z_3 & Z_4 & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}. \tag{3.4}$$

If all matrices $Z_i$ in (3.4) are nondefective, then the spectrum of the matrix $Q(z)$ consists of the eigenvalues of the $n$ matrices $Z_i$. This observation leads us to the condition of convergence

$$\rho(z_i D(I - z_i D)^{-1}(D^{-1}A - I)) < 1, \quad i = 1, \ldots, n,$$

where $\rho(\cdot)$ denotes the spectral radius function. This condition is identical to that of the PDIRK method. Constant stepsize plots of the convergence region $\mathbf{C} := \{z: \rho(zD(I - zD)^{-1}(D^{-1}A - I)) < 1\}$ for the Radau IIA correctors of orders $p = 3$, 5 and 7 reveal that the whole left halfplane is contained in $\mathbf{C}$. Hence, the Radau IIA based PDIRKAS J and PDIRKAS GS methods may be considered as "A-convergent". Thus, we may conclude that using these Radau IIA correctors leads to PDIRKAS methods whose stability region is completely determined by the stability region of the predictor.

### 3.2. *Rate of convergence*

Although the *regions* of convergence of the PDIRKAS and PDIRK methods are identical, the *rate* of convergence of the PDIRKAS method may be much worse because of ill-conditioning (or even defectiveness) of the eigensystem of the iteration matrix $Q(z)$. For example, if we integrate with fixed stepsizes, then $Q(z)$ possesses $s$ eigenvalues of geometric multiplicity $n$ leading to rather poor convergence as $n$ increases. The condition of the eigensystem may improve if all stepsizes are distinct, but convergence can still be slow.

In order to get insight into the convergence properties as a function of $j$ and $n$, we need an estimate for the rate of convergence of the iteration process. In this paper, we shall adopt a definition as given in [17, p. 88], where the *averaged rate of convergence* of the recursion (3.3) is given by

$$R(n, j, z) := -\log\left(\sqrt[j]{\| Q(z)^j \|}\right). \tag{3.5}$$

Let the iteration error associated with $\boldsymbol{Y}_i^{(j)}$, $i = 1, \ldots, n$, be of magnitude $10^{-\Delta(j)}$ (that is, the iterates $\boldsymbol{Y}_i^{(j)}$ and the corrector solutions $\boldsymbol{Y}_i$, $i \leqslant n$, differ by $\Delta(j)$ decimal digits). Then, taking logarithms to base 10, the number of iterations $j$ needed to achieve this is at most

$$j \approx 1 + \frac{\Delta(j) - \Delta(1)}{R(n, j-1, z)}. \tag{3.6}$$

We shall separately discuss the rate of convergence at the origin (*nonstiff* rate of

convergence), at infinity (*stiff* rate of convergence), and the rate of convergence at intermediate points in the whole left halfplane. At the origin, the matrices $Q_J$ and $Q_{GS}$ can be approximated by

$$Q_J(z) = K + \operatorname{diag}(z)L + O(z^2),$$

$$K := \begin{pmatrix} O & O & O & \cdots \\ E & O & O & \cdots \\ O & E & O & \cdots \\ O & O & E & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}, \quad L := \begin{pmatrix} A-D & O & O & O & \cdots \\ DE & A-D & O & O & \cdots \\ O & DE & A-D & O & \cdots \\ O & O & DE & A-D & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}, \tag{3.7a}$$

$$Q_{GS}(z) = M \operatorname{diag}(z) + O(z^2),$$

$$M := \begin{pmatrix} A-D & O & O & O & \cdots \\ H & A-D & O & O & \cdots \\ H & H & A-D & O & \cdots \\ H & H & H & A-D & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}, \quad H := E(A-D), \tag{3.7b}$$

and at infinity, we obtain

$$Q_J(z) = Q_{GS}(z)$$

$$:= \begin{pmatrix} I - D^{-1}A & O & O & O & \cdots \\ O & I - D^{-1}A & O & O & \cdots \\ O & O & I - D^{-1}A & O & \cdots \\ O & O & O & I - D^{-1}A & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} + O(z^{-1}). \tag{3.8}$$

In the following subsections, the maximum norm is used in the definition of $R(n, j, z)$, and in the tables of computed convergence rates, the underlying corrector is the four-stage Radau IIA method iterated by means of the matrix $D = D_4$ as defined in (2.3).

*3.2.1. Convergence of nonstiff error components*

From (3.7a) it follows that

$$[Q_J(z)]^j = [K + \operatorname{diag}(z)L + O(z^2)]^j = K^j + O(z). \tag{3.9}$$

Since $\| K^j \|_\infty$ equals 1 for $j < n$ and vanishes as $j \geqslant n$, we have that

$$\begin{aligned} R_J(n, j, z) &= O(z) && \text{for } j < n, \\ R_J(n, j, z) &= O\left(\frac{1}{j} |\log \| z \| |\right) && \text{for } j \geqslant n, \end{aligned} \tag{3.10a}$$

whereas (3.7b) immediately reveals that

$$R_{GS}(n, j, z) = O(|\log||z|||) \quad \text{for all } j. \tag{3.10b}$$

These formulas indicate that with respect to the nonstiff error components the convergence of Jacobi iteration is unacceptably slow. Therefore, in the remainder of this paper, we confine our discussions to the PDIRKAS GS method.

Let us consider convergence in more detail for *fixed* stepsizes, i.e. $z_i = z$ for all $i$. From (3.7b) it follows that

$$R_{GS}(n, j, z) = -\log|z| - \log\left(\sqrt[j]{||M^j||_\infty} + O(z)\right). \tag{3.11}$$

The following theorem provides explicit formulas for the asymptotic behaviour of the nonstiff rate of convergence for large values of $j$ and $n$, respectively.

**Theorem 3.2**

For fixed values of $n$, the nonstiff rate of convergence of the PDIRKAS GS method satisfies the asymptotic relation

$$R_{GS}(n, j, z) = -\log(\rho(A - D)|z|) - O(j^{-1}\log(j)) \quad \text{as } j \to \infty \quad \text{and} \quad z \to 0. \tag{3.12}$$

If the matrices $A$ and $D$ satisfy the conditions $a_{ss} < d_s < 1$ and $a_{sk} > 0$ $(k = 1, \ldots, s)$, then for fixed $j$

$$R_{GS}(n, j, z) = -\log(n|z|) - \log\left[(1 - d_s)\sqrt[j]{\frac{1 - 2a_{ss} + d_s}{j!(1 - d_s)}} + O(n^{-1})\right] \quad \text{as } n \to \infty \quad \text{and} \quad z \to 0. \tag{3.13}$$

*Proof*

The formula (3.12) immediately follows from the asymptotic formula for the norm of powers of matrices (see e.g. [17]). Assertion (3.13) can be proved along the lines of a similar theorem given in [13]. According to this proof, it is first shown that

$$||M^j||_\infty = ||M_0^j||_\infty + O(n^{j-1}) \quad \text{as } n \to \infty, \tag{3.14}$$

where

$$M_0 := \begin{pmatrix} O & O & O & \cdots & O \\ H & O & O & \cdots & O \\ H & H & O & \cdots & O \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ H & H & \cdots & H & O \end{pmatrix}, \quad H := E(A - D).$$

Next, it is shown that

$$||M_0^j||_\infty = \frac{1}{j!}n^j||H^j||_\infty + O(n^{j-1}) \quad \text{as } n \to \infty. \tag{3.15}$$

By observing that $H$ satisfies the recursion $H^j = (1 - d_s)^{j-1} H$, and using the assumptions $a_{sk} > 0$, $a_{ss} < d_s < 1$, we find

$$|| H^j ||_\infty = (1 - d_s)^{j-1}(1 - 2a_{ss} + d_s). \tag{3.16}$$

On substitution into (3.15) and into (3.14) formula (3.13) is immediate. □

If we consider the error over the whole integration interval, i.e. $n = N$, then this theorem shows that the nonstiff rate of convergence $R_{\mathrm{GS}}$ rapidly converges to a constant value as $N$ increases and $j$ is kept fixed. In this connection, we remark that the nonstiff rate of convergence of the PDIRK method is given by

$$R_{\mathrm{PDIRK}}(j, z) = -\log\left(\sqrt[j]{|| Z^j ||}\right) = -\log|z| - \log\left(\sqrt[j]{|| (A - D)^j ||_\infty} + O(z)\right), \tag{3.17}$$

showing that the nonstiff rate of convergence $R_{\mathrm{PDIRK}}$ behaves as $O(\log(N))$ as $N$ increases.

*3.2.2. Convergence of stiff error components*

If $z \to \infty$, then (3.8) immediately yields the following theorem:

**Theorem 3.3**

If $z \to \infty$ and if $n$ is finite, then for any corrector (2.1), the rate of convergence of the PDIRKAS GS method is given by

$$R_{\mathrm{GS}}(n, j, \infty) = -\frac{1}{j} \log || (I - D^{-1}A)^j ||_\infty. \tag{3.18}$$ □

We remark that the stiff rate of convergence of the PDIRKAS GS method is identical to that of the PDIRK method and does not depend on $n$. Table 1 lists the values for a few values of $j$.

*3.2.3. Convergence at intermediate values of z*

The preceding subsections indicate that the stiff and nonstiff rates of convergence of the PDIRKAS GS method are quite satisfactory, even for larger values of $n$. However, as soon as we move away from the origin or from infinity, then the rate of convergence deteriorates. Tables 2a and 2b respectively list values of Min $\{R_{\mathrm{GS}}(n, j, z): z \leqslant 0\}$ and Min $\{R_{\mathrm{GS}}(n, j, z): \mathrm{Re}(z) \leqslant 0\}$ for the four-stage Radau IIA corrector for a few values of $n$. In the latter case, the minimal rate of

Table 1
Stiff $R_{\mathrm{GS}}(n, j, \infty)$ values for the four-stage Radau IIA corrector.

| $j = 1$ | $j = 2$ | $j = 4$ | $j = 8$ | $j = 16$ | $j = 32$ | $j = \infty$ |
|---|---|---|---|---|---|---|
| −0.67 | −0.52 | 0.15 | 0.82 | 1.22 | 1.40 | 1.60 |

Table 2a
Values of Min $\{R_{GS}(n, j, z): z \leqslant 0\}$ and $j_\Delta$ for the four-stage Radau IIA corrector.

| $j$ | $n = 1$ | $n = 2$ | $n = 4$ | $n = 8$ |
|---|---|---|---|---|
| 1 | −0.67 | −0.67 | −0.67 | −0.67 |
| 2 | −0.52 | −0.52 | −0.52 | −0.54 |
| 4 | −0.01 | −0.25 | −0.36 | −0.43 |
| 8 | 0.33 | 0.09 | −0.16 | −0.33 |
| 16 | 0.52 | 0.35 | 0.12 | −0.12 |
| 32 | 0.60 | 0.51 | 0.35 | 0.14 |
| $j_\Delta(n)$ | 19 | 23 | 31 | 42 |
| $S(n)$ | 1 | 1.6 | 2.2 | 3.1 |

Table 2b
Values of Min $\{R_{GS}(n, j, z): \mathrm{Re}\,(z) \leqslant 0\}$ and $j_\Delta$ for the four-stage Radau IIA corrector.

| j | $n = 1$ | $n = 2$ | $n = 4$ | $n = 8$ |
|---|---|---|---|---|
| 1 | −0.67 | −0.81 | −0.94 | −1.05 |
| 2 | −0.52 | −0.60 | −0.75 | −0.90 |
| 4 | −0.14 | −0.38 | −0.57 | −0.75 |
| 8 | 0.09 | −0.12 | −0.35 | −0.45 |
| 16 | 0.18 | 0.02 | −0.13 | −0.29 |
| 32 | 0.23 | 0.15 | 0.03 | −0.11 |
| 64 | 0.25 | 0.21 | 0.14 | 0.02 |
| $j_\Delta(n)$ | 42 | 52 | 69 | 102 |
| $S(n)$ | 1 | 1.6 | 2.3 | 3.1 |

convergence was always found on the imaginary axis. Tables 2 show that for larger values of $n$, the rate of convergence only becomes positive if the number of iterations is relatively large, particularly in the case of table 2b. The effect on the corresponding iteration error components is disastrous (even for relatively low values of $n$), because these components start to grow exponentially and will only be damped if $j$ is relatively large. In order to illustrate this, tables 2a and 2b also list the values of $j = j_\Delta(n)$ given by (3.6) with $\Delta - \Delta_1 = 10$ and $z = ze$. The rather large values of $j_\Delta(n)$ as $n$ increases indicate that the iterates may easily become so bad that we have overflow before the iteration process starts to converge. Therefore, some strategy should be employed that controls when it is safe to advance to the next step point (see section 4.3). Finally, we remark that by means of the values of $j_\Delta$ we can compute an estimate of the speed-up factor of PDIRKAS GS with respect to PDIRK. Setting $n = N$, the number of sequential iterations of these methods are respectively given by $N + j_\Delta(N) - 1$ and $Nj_\Delta(1)$, resulting in the speed-up factor $S(N) = Nj_\Delta(1)[N + j_\Delta(N) - 1]^{-1}$ (notice that the speed-up factors along the negative and imaginary axis are roughly equal).

## 4. Numerical experiments

The PDIRKAS GS method {(2.2), (2.5)} described above was applied using the four-stage Radau IIA corrector equation and the predictor formula (2.7). Since the number $m(t_n)$ of outer iterations needed to solve the corrector equation will strongly depend on $n$, we applied a dynamic iteration strategy with stopping criterium (cf. [13])

$$\Delta_n^{(j)} = \frac{\| (\boldsymbol{e}_s^{\mathrm{T}} E \otimes I)(\boldsymbol{Y}_n^{(j-1)} - \boldsymbol{Y}_n^{(j)}) \|_1}{\| (\boldsymbol{e}_s^{\mathrm{T}} E \otimes I) \boldsymbol{Y}_n^{(j-1)} \|_1} \leqslant TOL_{\mathrm{corr}}.$$

In all our experiments, we set $TOL_{\mathrm{corr}} = 10^{-12}$. The number of necessary processors is determined by the number of step points at which this stopping criterion is not yet satisfied. The maximal number of processors needed during the integration equals $sK_{\max}$, where $K_{\max}$ denotes the maximal number of step points where the stopping criterion is not yet satisfied. For the inner iteration process for solving the correction formula in (2.2) we used a modified Newton method which was solved to convergence.

In addition to the PDIRKAS GS method, we shall also apply the PDIRK method that may be considered as the PDIRKAS GS method in one-processor mode. In this paper, we want to compare characteristic properties of the methods like the rate of convergence and sequential costs, rather than strategy aspects such as stepsize and error control. Therefore, we restrict the experiments to problems that can be integrated with fixed stepsizes $h = N^{-1}T$. In a sequel to this paper, we will develop a stepsize and error control strategy [18].

The calculations were performed using 15-digits arithmetic. The accuracy is given by the number of correct digits $\Delta$, obtained by writing the maximum norm of the absolute error at the endpoint in the form $10^{-\Delta}$.

### 4.1. *Test problems*

Our first problem is the well known stability test problem of Prothero and Robinson

$$\frac{dy}{dt} = -\epsilon^{-1}(y - g(t)) + g'(t), \quad y(0) = g(0), \quad 0 \leqslant t \leqslant T, \tag{4.1a}$$

where the exact solution equals $g(t)$ and $\epsilon$ is a small parameter. Prothero and Robinson used this problem to show the order reduction of RK methods when $\epsilon$ is small. In our experiments we set

$$g(t) = \cos(t), \quad \epsilon = 10^{-3}. \tag{4.1b}$$

The second test problem is the "nonlinearization" of problem (4.1):

$$\frac{dy}{dt} = -\epsilon^{-1}(y^3 - g(t)^3) + g'(t), \quad y(0) = g(0), \quad 0 \leqslant t \leqslant T, \tag{4.2a}$$

with exact solution $y(t) = g(t)$ for all values of the parameter $\epsilon$. As in the preceding problem, we set

$$g(t) = \cos(t), \quad \epsilon = 10^{-3}. \tag{4.2b}$$

The third test problem is that of Kaps [14]:

$$\begin{aligned} \frac{dy_1}{dt} &= -(2 + \epsilon^{-1})y_1 + \epsilon^{-1}(y_2)^2, \\ \frac{dy_2}{dt} &= y_1 - y_2(1 + y_2), \\ y_1(0) &= y_2(0) = 1, \quad 0 \leqslant t \leqslant T, \end{aligned} \tag{4.3}$$

with the smooth exact solution $y_1 = \exp(-2t)$ and $y_2 = \exp(-t)$ for all values of the parameter $\epsilon$. This problem belongs to the class of problems for which stiffly accurate RK methods do not suffer order reduction whatever small $\epsilon$ is (cf. [9]).

The test set of Enright et al. [7] contains the following system of ODEs describing a chemical reaction:

$$\frac{d\boldsymbol{y}}{dt} = -\begin{pmatrix} 0.013 + 1000y_3 & 0 & 0 \\ 0 & 2500y_3 & 0 \\ 0.013 & 0 & 1000y_1 + 2500y_2 \end{pmatrix}\boldsymbol{y}, \tag{4.4a}$$

with $\boldsymbol{y}(0) = (1, 1, 0)^{\mathrm{T}}$. Since we want to use fixed step sizes in our experiments, we avoided the initial phase by choosing the starting point at $t_0 = 1$. The corresponding initial and end point values at $t = T = 51$ are given by

$$\boldsymbol{y}(1) \approx \begin{pmatrix} 0.990731920827 \\ 1.009264413846 \\ -0.366532612659 \times 10^{-5} \end{pmatrix}, \quad \boldsymbol{y}(51) \approx \begin{pmatrix} 0.591045966680 \\ 1.408952165382 \\ -0.186793736719 \times 10^{-5} \end{pmatrix}. \tag{4.4b}$$

The final test example is taken from Lambert [15, p. 228]:

$$\frac{d\boldsymbol{y}}{dt} = \begin{pmatrix} 42.2 & 50.1 & -42.1 \\ -66.1 & -58.0 & 58.1 \\ 26.1 & 42.1 & -34.0 \end{pmatrix}\boldsymbol{y}, \tag{4.5a}$$

with $\boldsymbol{y}(0) = (1, 0, 2)^{\mathrm{T}}$. As soon as the fast transient $e^{-50t}$ has died out, the exact solution is sinusoidal with a slowly increasing amplitude:

$$\boldsymbol{y}(t) = \begin{pmatrix} e^{t/10}\sin(8t) + e^{-50t} \\ e^{t/10}\cos(8t) - e^{-50t} \\ e^{t/10}(\sin(8t) + \cos(8t)) + e^{-50t} \end{pmatrix}. \tag{4.5b}$$

The eigenvalues of this system are given by $-50$ and $1/10 \pm 8i$, hence we are faced with a stiff problem, the nonstiff solution components of which are nondissipative.

As pointed out in Lambert's discussion of the system (4.5), such problems are a difficult test for $L$-stable methods like our Radau IIA based PDIRKAS GS method. As in the preceding example, the initial phase is avoided by starting the integration at $t_0 > 0$. In fact, we integrated the interval [0.5, 1.5].

### 4.2. *Comparison of the PDIRKAS GS and the PDIRK method*

In our first tests, we compare results obtained by the PDIRKAS GS and the PDIRK method. We apply the PDIRKAS GS in unlimited-number-of-processors mode and in one-processor mode (by which we generate the PDIRK method). The sequential computational complexity is measured by the total number $N_{\text{seq}} = N - 1 + m(t_N)$ of sequential implicit systems to be solved during the integration process. Furthermore, we define the average number of iterations per step and the average number of *sequential* iterations per step by $m^* := N^{-1}\Sigma_n m(t_n)$ and

Table 3
Results for the linear Prothero–Robinson problem (4.1) with $T = 1$.

| $N$ | $\Delta$ | $K_{\max}$ | $m^*$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 1 | 6.3 | 1 | 10.0 | 10 | 10.0 | 1.0 |
| 2 | 7.4 | 2 | 10.5 | 13 | 6.5 | 1.5 |
| 4 | 8.6 | 4 | 12.8 | 19 | 4.8 | 2.2 |
| 8 | 9.8 | 8 | 17.3 | 32 | 4.0 | 2.8 |
| 16 | 11.0 | 13 | 26.9 | 59 | 3.7 | 3.1 |

Table 4
Results for the nonlinear Prothero–Robinson problem (4.2) with $T = 1$.

| $N$ | $\Delta$ | $K_{\max}$ | $m^*$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 1 | 6.3 | 1 | 10.0 | 10 | 10.0 | 1.0 |
| 2 | 7.3 | 2 | 9.5 | 12 | 6.0 | 1.6 |
| 4 | 8.5 | 4 | 11.3 | 18 | 4.5 | 2.1 |
| 8 | 9.7 | 7 | 15.4 | 28 | 3.5 | 2.9 |
| 16 | 11.0 | 13 | 22.7 | 53 | 3.3 | 3.2 |

Table 5a
Results for the Kaps problem (4.3) with $\epsilon = 10^{-3}$ and $T = 1$.

| $N$ | $\Delta$ | $K_{\max}$ | $m^*$ | $N_{\text{seq}}$ | $m^*_{\text{seq}}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 1 | 5.0 | 1 | 12.0 | 12 | 12.0 | 1.0 |
| 2 | 6.4 | 2 | 13.0 | 15 | 7.5 | 1.7 |
| 4 | 7.8 | 4 | 15.3 | 22 | 5.5 | 2.3 |
| 8 | 9.1 | 8 | 20.9 | 36 | 4.5 | 2.8 |
| 16 | 10.3 | 14 | 31.4 | 64 | 4.0 | 3.4 |

Table 5b
Results for the Kaps problem (4.3) with $\epsilon = 10^{-8}$ and $T = 1$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 1 | 6.6 | 1 | 12.0 | 12 | 12.0 | 1.0 |
| 2 | 8.7 | 2 | 10.5 | 13 | 6.5 | 1.6 |
| 4 | 10.8 | 4 | 9.3 | 14 | 3.5 | 2.6 |

Table 6
Results for the chemical reaction problem (4.4).

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 1 | 7.9 | 1 | 9.0 | 9 | 9.0 | 1.0 |
| 2 | 9.8 | 2 | 7.5 | 10 | 5.0 | 1.5 |
| 4 | 11.8 | 4 | 7.0 | 11 | 2.8 | 2.5 |

Table 7
Results for the nondissipative problem (4.5) with $T = 1.5$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 5.9 | 10 | 18.5 | 33 | 3.3 | 4.7 |
| 20 | 8.1 | 14 | 20.7 | 53 | 2.6 | 4.6 |
| 40 | 10.2 | 21 | 25.1 | 85 | 2.1 | 4.6 |
| 80 | 12.3 | 33 | 30.4 | 138 | 1.7 | 4.8 |

$m^*_{seq} := N^{-1}N_{seq}$, respectively. For the PDIRK method, we obviously have $N_{seq} = \Sigma_n m(t_n)$ and $m^* = m^*_{seq} = N^{-1}N_{seq}$. The ratio of the values of $m^*_{seq}$ for the PDIRKAS GS and PDIRK methods determines the speed-up factor $S(N)$.

Tables 3 to 7 present multi-processor results for the PDIRKAS GS method and the speed-up factors $S(N)$. From these results, we conclude that the PDIRKAS GS method becomes more efficient as the number of step points $N$ increases and that the speed-up factors $S(N)$ are in good agreement with the theoretical speed-up factors listed in tables 2a and 2b.

In order to see the effect of larger intervals of integration, we repeated the experiments for the linear Prothero–Robinson problem (4.1) and the Kaps problem (4.3), but now on the interval [0, 10]. The results in tables 8 and 9 reveal that the speed-up factor is much larger than in tables 3 and 5 (for the same stepsize) with a maximal speed-up factor of about 5, but we also see that the convergence behaviour in the Prothero–Robinson problem and the mildly stiff Kaps problem now becomes worse as $N$ becomes too large. This is due to the deterioration of the rate of convergence as discussed in subsection 3.2.3. For the Kaps problem with $\epsilon = 10^{-8}$, table 9b shows that this deterioration does not occur and hence the limiting value $m^*_{seq} = 1$ is almost obtained.

Table 8
Results for the linear Prothero–Robinson problem (4.1) with $T = 10$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 6.9 | 9 | 17.2 | 31 | 3.1 | 3.6 |
| 20 | 7.7 | 16 | 22.4 | 46 | 2.3 | 4.7 |
| 40 | 8.7 | 29 | 32.6 | 77 | 1.9 | 5.7 |
| 80 | 10.0 | 55 | 69.2 | 172 | 2.1 | 5.2 |
| 160 | | ... | divergence | | ... | |

Table 9a
Results for the Kaps problem (4.3) with $\epsilon = 10^{-3}$ and $T = 10$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 9.5 | 10 | 22.0 | 39 | 3.9 | 4.1 |
| 20 | 11.6 | 17 | 30.6 | 65 | 3.3 | 3.9 |
| 40 | 13.7 | 30 | 47.5 | 110 | 2.8 | 4.3 |
| 80 | 15.8 | 57 | 92.6 | 245 | 3.1 | 3.8 |
| 160 | | ... | divergence | | ... | |

Table 9b
Results for the Kaps problem (4.3) with $\epsilon = 10^{-8}$ and $T = 10$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 9.5 | 10 | 20.3 | 36 | 3.6 | 4.5 |
| 20 | 11.6 | 15 | 20.1 | 49 | 2.5 | 5.1 |
| 40 | 13.7 | 21 | 22.7 | 75 | 1.9 | 5.6 |
| 80 | 16.0 | 28 | 24.3 | 117 | 1.5 | 5.9 |
| 160 | 17.5 | 33 | 24.8 | 198 | 1.2 | 6.4 |

### 4.3. *Dynamic PDIRKAS GS method*

The preceding experiments indicate that the performance of the PDIRKAS GS method strongly depends on the problem to be solved. Therefore, we should apply a strategy that controls when it is safe to move to the next time point $t_n$. One strategy is to take the local truncation error of the last component of the iterate $\boldsymbol{Y}_{n-1}^{(j)}$ as a measure for safety (in fact, such a strategy was used in [13] for *nonstiff* problems). However, in the present case of *stiff* problems, the BDF predictor formula (2.7) often computes highly accurate first iterates $\boldsymbol{Y}_n^{(1)}$ for all $n$, so that control of its local truncation error will not be effective. The cause of a potential bad performance is a strong initial grow of the iteration error. Hence, an alternative strategy might be a check on the behaviour of the iteration error, e.g. by means of the residue of the corrector equation (2.1a). If this residue does not grow anymore, then it should be safe to advance to the next step point.

Let us define the residual function

$$\boldsymbol{R}_n^{(j)} := \boldsymbol{Y}_n^{(j)} - (E \otimes I_d)\boldsymbol{Y}_{n-1}^{*} - h_n(A \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(j)}), \tag{4.6}$$

where $\boldsymbol{Y}_{n-1}^{*}$ denotes the most recent iterate available at $t_{n-1}$ (this iterate will depend on $j$ too). One option is to require that for some iteration index $i$

$$\| \boldsymbol{e}_s^{\mathrm{T}} \boldsymbol{R}_{n-k}^{(i)} \|_\infty < a \| \boldsymbol{e}_s^{\mathrm{T}} \boldsymbol{R}_{n-k}^{(1)} \|_\infty, \quad i \geqslant 2, \quad k \geqslant 1, \quad a \leqslant 1, \tag{4.7}$$

before advancing to $t_n$. Here, $a$ is some safety parameter and $k$ determines the point where the iteration errors are checked. Since the first few iterations at $t_{n-k}$ may have an erratic behaviour, we should choose $k$ greater than 1. Setting $j^*(t_{n-1})$ equal to the current iteration index $j$ at $t_{n-1}$, as soon as the residue at $t_{n-k}$ is sufficiently small, we can compute the iterate $\boldsymbol{Y}_n^{(j)}$ according to the formula

$$\boldsymbol{Y}_n^{(j)} - h_n(D \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(j)}) = (E \otimes I_d)\boldsymbol{Y}_{n-1}^{(j+j^*-1)} + h_n((A - D) \otimes I_d)\boldsymbol{F}(\boldsymbol{Y}_n^{(j-1)}),$$
$$j = 2, \ldots, m(t_n). \tag{4.8}$$

The sequential costs are now proportional to the number

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

Tables 10, 11 and 12 are the analogues of tables 8, 9 and 7, respectively. By virtue of the strategy (4.7), using $a = 10^{-2}$ and $k = 3$, divergence of the iteration process is avoided at the costs of a modest increase of the sequential costs. However, $K_{\max}$ is also much lower, which decreases the number of processors substantially.

Table 10
Results for the linear Prothero–Robinson problem (4.1) with $T = 10$.

| $N$ | $\Delta$ | $K_{\max}$ | $m^*$ | $N_{\mathrm{seq}}$ | $m^*_{\mathrm{seq}}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 6.9 | 7 | 14.3 | 31 | 3.1 | 3.6 |
| 20 | 7.6 | 8 | 16.0 | 55 | 2.8 | 3.9 |
| 40 | 8.8 | 8 | 17.5 | 108 | 2.7 | 3.9 |
| 80 | 10.0 | 8 | 19.3 | 230 | 2.9 | 3.8 |
| 160 | 11.3 | 8 | 19.8 | 513 | 3.2 | 3.6 |

Table 11a
Results for the Kaps problem (4.3) with $\epsilon = 10^{-3}$ and $T = 10$.

| $N$ | $\Delta$ | $K_{\max}$ | $m^*$ | $N_{\mathrm{seq}}$ | $m^*_{\mathrm{seq}}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 9.5 | 7 | 18.4 | 39 | 3.9 | 4.1 |
| 20 | 11.6 | 10 | 20.7 | 65 | 3.3 | 3.9 |
| 40 | 13.7 | 14 | 23.2 | 116 | 2.9 | 4.2 |
| 80 | 15.8 | 17 | 25.4 | 248 | 3.1 | 3.8 |
| 160 | 17.7 | 9 | 23.8 | 532 | 3.3 | 3.6 |

Table 11b
Results for the Kaps problem (4.3) with $\epsilon = 10^{-8}$ and $T = 10$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 9.5 | 8 | 17.8 | 36 | 3.6 | 4.5 |
| 20 | 11.6 | 7 | 13.8 | 49 | 2.5 | 5.1 |
| 40 | 13.7 | 9 | 13.0 | 76 | 1.9 | 5.3 |
| 80 | 15.8 | 9 | 10.5 | 127 | 1.6 | 5.0 |
| 160 | 16.9 | 10 | 8.9 | 233 | 1.5 | 5.1 |

Table 12
Results for the nondissipative problem (4.5) with $T = 1.5$.

| $N$ | $\Delta$ | $K_{max}$ | $m^*$ | $N_{seq}$ | $m^*_{seq}$ | $S(N)$ |
|---|---|---|---|---|---|---|
| 10 | 5.9 | 8 | 16.9 | 34 | 3.4 | 4.5 |
| 20 | 8.1 | 10 | 16.9 | 54 | 2.7 | 4.5 |
| 40 | 10.2 | 15 | 19.0 | 85 | 2.1 | 4.6 |
| 80 | 12.3 | 20 | 20.6 | 138 | 1.7 | 4.8 |

Summarizing, we may conclude that the PDIRKAS GS method using the strategy defined by (4.7) is rather robust. As to the sequential costs, it follows from the tables of results, that the effective number of iterations per step for solving the four-stage Radau IIA RK corrector varies from 1.5 to at most 4 iterations per step. With respect to the PDIRK method, the speed-up factors are in the range 3.5 until 5. Taking into account that the variable step version of the PDIRK method (viz. the PSODE code described in [16]) is about twice as fast as the best sequential codes such as LSODE, we may expect that the variable step version of the PDIRKAS GS method will give rise to speed-up factors in the range 7 until 10 with respect to LSODE. This variable step version will be discussed in a future paper [18].

## References

[1] A. Bellen, Parallelism across the steps for difference and differential equations, in: *Numerical Methods for Ordinary Differential Equations*, Lecture Notes in Mathematics 1386 (Springer, 1987) pp. 22–35.

[2] A. Bellen, R. Vermiglio and M. Zennaro, Parallel ODE-solvers with stepsize control, J. Comp. Appl. Math. 31 (1990) 277–293.

[3] K. Burrage, The search for the Holy Grail, or Predictor–Corrector methods for solving ODEIVPs, Appl. Numer. Math. 11 (1993) 125–141.

[4] J.C. Butcher, On the convergence of numerical solutions to ordinary differential equations, Math. Comp. 20 (1966) 1–10.

[5] J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations, Runge–Kutta and General Linear Methods* (Wiley, Chichester/New York/Brisbane/Toronto/Singapore, 1987).

[6] P. Chartier, Parallelism in the numerical solution of initial value problems for ODEs and DAEs, Thesis, Université de Rennes I, France (1993).
[7] W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of ODEs, BIT 15 (1975) 10–48.
[8] C.W. Gear and K.W. Tu, The effect of variable mesh size on the stability of multistep methods, SIAM J. Numer. Anal. 11 (1974) 1025–1043.
[9] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations, II: Stiff and Differential Algebraic Problems* (Springer, Berlin, 1991).
[10] P.J. van der Houwen and B.P. Sommeijer, Iterated Runge–Kutta methods on parallel computers, SIAM J. Sci. Stat. Comp. 12 (1991) 1000–1028.
[11] P.J. van der Houwen and B.P. Sommeijer, Preconditioning in parallel Runge–Kutta methods for stiff initial value problems, to appear in CMA (1994).
[12] P.J. van der Houwen, B.P. Sommeijer and W. Couzy, Embedded diagonally implicit Runge–Kutta algorithms on parallel computers, Math. Comp. 58 (1992) 135–159.
[13] P.J. van der Houwen, B.P. Sommeijer and W.A. van der Veen, Parallel iteration across the steps of high order Runge–Kutta methods for nonstiff initial value problems, to appear in JCAM (1994).
[14] P. Kaps, Rosenbrock-type methods, in: *Numerical Methods for Stiff Initial Value Problems*, eds. G. Dahlquist and R. Jeltsch, Bericht nr. 9, Inst. für Geometrie und Praktische Mathematik der RWTH Aachen (1981).
[15] J.D. Lambert, *Numerical Methods for Ordinary Differential Equations*, (Wiley, New York, 1991).
[16] B.P. Sommeijer, Parallelism in the numerical integration of initial value problems, Thesis, University of Amsterdam (1992).
[17] D.M. Young, *Iterative Solution of Large Linear Systems* (Academic Press, New York, 1971).
[18] W.A. van der Veen, Performance of step-parallelism in Runge–Kutta methods for stiff initial-value problems, in preparation (1994).

# Chapter 4

# Convergence Aspects of Step-Parallel Iteration of Runge-Kutta Methods

W.A. van der Veen, J.J.B. de Swart, P.J. van der Houwen

**Abstract**

One of the most powerful methods for solving initial value problems for ordinary differential equations is an implicit Runge-Kutta method such as the Radau IIA methods. These methods are both highly accurate and highly stable. However, the iterative scheme needed for solving the implicit RK equations requires a lot of computational effort. The arrival of parallel computer systems has changed the situation in the sense that the effective computational effort can be reduced to a large extent. One option is the application of the iteration scheme concurrently at a number of step points on the $t$-axis. In this paper, we shall analyse the convergence of a special class of such step-parallel iteration methods.

*CR Subject Classification (1991):* G.1.7
*Keywords & Phrases:* numerical analysis, Runge-Kutta methods, parallelism, convergence factors.
*Note:* The research reported in this paper was partly supported by the Technology Foundation (STW) in the Netherlands.

## 1 Introduction

We consider parallel methods for solving $d$-dimensional initial value problems (IVPs):

$$y'(t) = f(y(t)), \quad y(t_0) = y_0, \quad y, f \in \mathbf{R}^d. \tag{1}$$

One of the most powerful methods for solving this IVP is an implicit Runge-Kutta (RK) method such as the Radau IIA methods. These methods are L-stable and have order $p = 2s - 1$, $s$ being the number of stages. However, the iterative scheme needed for solving the implicit RK equations requires a lot of computational effort. Because of this, implicit RK methods have never been

popular on sequential computers. Parallel computer systems have changed the situation, and various attempts have been made to develop parallel iteration schemes for solving the implicit RK equations. We mention the work of Jackson and Nørsett [9], Lie [10], Bellen et al. [1], [2] and Chartier [4]. Also at CWI, parallel iteration schemes have been investigated. For stiff problems, we applied Newton-type iteration in which the $sd \times sd$ Jacobian matrix of the implicit equations was approximated by a block diagonal matrix with $d \times d$ blocks (cf. [6]). The sequential (or effective) costs per iteration of the resulting 'simplified' Newton iteration method are reduced to solving $s$ linear systems of dimension $d$ in parallel. This iteration method was called the PDIRK iteration method (Parallel Diagonal-implicit Iterated RK method). Following the ideas of Bellen and co-workers, a further level of parallelism was introduced in [8], [7] and [12] by applying the PDIRK iteration scheme concurrently at a number of step points on the $t$-axis. In this paper, we shall analyse the convergence of these step-parallel PDIRK methods.

## 2 The iteration scheme

Our starting point is the same corrector formula as in [7]. Using the General Linear Method notation of Butcher, the corrector formula reads (cf. [3] or [5])

$$Y_n = (E \otimes I)Y_{n-1} + h_n(A \otimes I)F(Y_n), \quad n = 1, \ldots, N. \tag{2}$$

Here, $h_n$ denotes the stepsize $t_n - t_{n-1}$, the $s \times s$ matrices $A$ and $E$ contain the method parameters, and $F(Y_n)$ contains the derivative values $(f(Y_{n,i}))$, where $Y_{n,i}$, $i = 1, 2, \ldots, s$, denote the $d$-dimensional components of the stage vector $Y_n$. In this paper we will assume that (2) possesses $s$ implicit stages and that the last stage corresponds to the step point $t_n$ (e.g. Radau IIA type methods). The $s$ components $Y_{n,i}$ represent numerical approximations at the intermediate points $t_{n-1} + c_i h_n$, $i = 1, \ldots, s$, where $c = (c_i) = Ae$, $e$ being the vector with unit entries. Furthermore, the matrix $I$ is the $d \times d$ identity matrix, $\otimes$ denotes the Kronecker product, and we define $Y_0 = e \otimes y_0$. The dimensions of $I$ and $e$ may change, but will always be clear from the context.

Confining our considerations to RK methods, the matrix $E$ in (2) is of the form

$$E := \begin{pmatrix} 0 & \ldots & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 \end{pmatrix}. \tag{3}$$

However, most of our analysis applies to the case of a General Linear Method where $E$ is more general.

We approximate the solution $Y_n$ of {(2),(3)} by successive iterates $Y_n^{(j)}$ satisfying the iteration scheme

$$\begin{aligned} &Y_n^{(1)} \text{ to be defined by the predictor formula,} \\ &Y_n^{(j)} - h_n(B \otimes I)F(Y_n^{(j)}) = (E \otimes I)Y_{n-1}^{(j+j^*-1)} \\ &\qquad\qquad +h_n((A-B) \otimes I)F(Y_n^{(j-1)}),\ j = 2, \ldots, m, \\ &Y_n^{(j)} = Y_n^{(m)},\ j > m, \end{aligned} \tag{4}$$

where $n = 1, 2, \ldots, N$, $B$ is an $s \times s$ matrix, $Y_0^{(j)} = e \otimes y_0$ for all $j$, and $j^*$ is an integer greater than or equal to 1. It will be assumed that the sequential costs of applying the predictor formula and the correction formula are comparable.

Irrespective the choice of the matrix $B$, the iteration scheme (4) possesses *parallelism across the steps*. For instance, if $j^*$ is constant, then (4) shows that for a given $j \geq 1$, the iterates $\{Y_n^{(j)}, Y_{n-1}^{(j+j^*)}, Y_{n-2}^{(j+2j^*)}, \ldots, Y_1^{(j+nj^*-j^*)}\}$ can be computed concurrently. The sequential (or effective) costs consists of $N_{\text{seq}} := m + (N-1)j^*$ applications of the correction formula (if $m$ depends on the step number $n$, then $m$ is understood to be the number of iterations at the endpoint). Notice that for $j^* = m$, the iteration method (4) reduces to the conventional iteration strategy without step parallelism.

The matrix $B$ defines the iteration method within a single step and therefore plays a crucial role in the degree of parallelism within the steps. There are several options for choosing the matrix $B$. For example, the case $B = 0$ (fixed point iteration) was studied in [8] and the resulting method was called the PIRKAS method (Parallel Iterated RK Across the Steps). In addition to parallelism across the *steps*, PIRKAS methods also have parallelism across the *components of the iterates*, because all components of $F(Y_n^{(j-1)})$ can also be evaluated in parallel. Methods where $B$ is a diagonal matrix $D$ with positive diagonal entries minimizing the spectral radius of the matrix $I - D^{-1}A$ (such matrices can be found in [6]) were applied in [7] and were called PDIRKAS methods (Parallel Diagonal-implicitly Iterated RK Across the Steps). These methods are implicit because we have to solve nonlinear relations in each iteration. But the diagonal structure of $B$ enables us to solve the components of $Y_n^{(j)}$ in parallel. Hence, we again have both parallelism across the *steps* and across the *components of the iterates*.

In an actual implementation, the number of iterations $m$ performed at $t_n$ and the parameter $j^*$ are defined dynamically. The value of $m$ is determined by the condition that for $j = m$ the iterates $Y_n^{(j)}$ satisfy the corrector equation (2) within a given tolerance. The value of $j^*$ turns out to be decisive for the overall performance of the iteration process. It should be sufficiently large in order to have satisfactory convergence at $t_n$. Hence, both $m$ and $j^*$ may depend on $t_n$. In a theoretical analysis, however, it seems not feasible to allow the parameters

$m$ and $j^*$ to be arbitrary functions of $n$, so that in deriving convergence results, $m$ and $j^*$ are assumed to be constant. In our first investigations of step-parallel iterations schemes in [8], [7], we hoped that sufficient robustness could already be obtained for $j^* = 1$. We therefore analysed convergence only for $j^* = 1$. However, our numerical experiments have shown that $j^*$ is at best 2 or 3. In this paper, we extend our earlier analysis to the case where $j^*$ is allowed to be greater than 1.

# 3 Stability and convergence

Assuming that the corrector equation (2) is unconditionally stable and that the corrector equation is solved within a given tolerance, the method (4) will be stable whenever it is convergent. We shall discuss convergence for the familiar basic test equation $y'(t) = \lambda y(t)$, where $\lambda$ is assumed to run through the spectrum of $\partial f/\partial y$. Furthermore, we assume $h$, $m$ and $j^*$ independent of $n$. When applied to the test equation, the iteration scheme assumes the form

$$Y_n^{(j)} = \begin{cases} \text{to be defined by the predictor formula,} & j = 1, \\ KY_{n-1}^{(j+j^*-1)} + ZY_n^{(j-1)}, & j = 2, \ldots, m, \\ Y_n^{(m)}, & j > m, \end{cases} \tag{5}$$

where

$$K := (I - zB)^{-1}E, \quad Z := z(I - zB)^{-1}(A - B), \quad z := \lambda h.$$

In [8], [7] we discussed convergence of (5) for the case $j^* = 1$. In this paper, we shall allow $j^*$ to be greater than 1. As already observed in [8], the convergence analysis of (4) cannot be restricted to a local analysis of the iteration errors at a fixed point $t_n$, but should be a global analysis where iteration errors at all preceding step points are involved. We shall distinguish two situations: $(i)$ the predictor is based on iterates generated by the iteration scheme (5), and $(ii)$ the iterates $Y_n^{(1)}$ are generated independently, that is, the predictor is completely independent of the iteration scheme. In the first situation, it is required that the predictor formula is explicitly given (to be referred to as the *given-predictor* case). In the second case, the predictor formula itself is not used in deriving the convergence conditions and may therefore have any form (the *independent-predictor* case). However, in the case of large integration intervals where $n$ becomes large, the predictor formula should be sufficiently stable in order to generate useful first iterates. In fact, for large $n$, the region of convergence of (5) will be limited by the stability region of the predictor. In

the given-predictor case, we confine our considerations to predictor formulas of the form

$$Y_n^{(1)} = P(Y_{n-1}^{(j^*)}). \tag{6}$$

For the test equation, the predictor formula (6) takes the form $Y_n^{(1)} = PY_{n-1}^{(j^*)}$, where $P = P(z)$ is an $s \times s$ matrix, to be called the *predictor matrix*. Thus, the step-parallel iteration method can be characterized by the matrices $K$, $Z$ and $P$ (if the predictor formula is explicitly specified).

In order to analyse convergence, we derive a relation between the vectors of iterates at $t_n$ and $t_{n-1}$. Repeated application of the recursion (5) yields

$$Y_n^{(j)} = \begin{cases} \sum_{k=1}^{j-1} Z^{k-1} K Y_{n-1}^{(j+j^*-k)} + Z^{j-1} Y_n^{(1)}, & j = 2, \ldots, m, \\ Y_n^{(m)}, & j > m. \end{cases} \tag{7}$$

Let $\theta = 0$ and $\theta = 1$ respectively refer to the independent-predictor and given-predictor cases introduced above. Then the recursion (7) can be written in the compact form

$$V_n = SV_{n-1} + (1-\theta)CY_n^{(1)}, \quad V_n := \begin{pmatrix} Y_n^{(j^*)} \\ Y_n^{(j^*+1)} \\ Y_n^{(j^*+2)} \\ \vdots \\ Y_n^{(m+j^*-1)} \end{pmatrix}, \quad C := \begin{pmatrix} Z^{j^*-1} \\ Z^{j^*} \\ \vdots \\ Z^{m-1} \\ \vdots \\ Z^{m-1} \end{pmatrix}, \tag{8}$$

$$S := \begin{pmatrix} \theta Z^{j^*-1}P & Z^{j^*-2}K & Z^{j^*-3}K & \ldots & K & O & \ldots & \ldots & O \\ \theta Z^{j^*}P & Z^{j^*-1}K & Z^{j^*-2}K & \ldots & . & K & O & \ldots & O \\ \vdots & \vdots & \vdots & \ldots & & & \ddots & \ddots & \vdots \\ \theta Z^{m-2}P & Z^{m-3}K & Z^{m-4}K & \ldots & & & \ldots & K & 0 \\ \theta Z^{m-1}P & Z^{m-2}K & Z^{m-3}K & \ldots & & & & \ldots & K \\ \vdots & \vdots & \vdots & \ldots & & & & & \vdots \\ \theta Z^{m-1}P & Z^{m-2}K & Z^{m-3}K & \ldots & & & & \ldots & K \end{pmatrix}.$$

$S$ and $C$ are an $m \times m$ and an $m \times 1$ block matrix, respectively. In both matrices, the last $j^*$ block rows are identical. If $j^* = 1$, then the first block row of $S$ reduces to $(\theta P \;\; O \;\; \ldots \;\; O)$, so that $S$ becomes a lower triangular block matrix.

If $\theta = 0$, then the predictor matrix $P$ is ignored in (8). Instead, the predictor values $Y_n^{(1)}$, $n = 1, 2, \ldots$, are involved. These values may be any sequence of initial iterates. If $\theta = 1$, then the special form of the predictor formula $Y_n^{(1)} = PY_{n-1}^{(j^*)}$ is taken into account.

## 3.1 The iteration error

In the conventional iteration process where $j^* = m$, we can derive a relation between the iteration error $\varepsilon_n^{(j)} := Y_n^{(j)} - Y_n$ and $\varepsilon_n^{(j-1)}$. However, if $j^* < m$, then this is no longer possible. In [8], [7] it was shown that if, and only if, $j^* = 1$, then there exists a relation between the set of iteration errors $\varepsilon^{(j)} := (\varepsilon_1^{(j)}, \varepsilon_2^{(j)}, \ldots, \varepsilon_n^{(j)})$ and $\varepsilon^{(j-1)}$. For $j^* > 1$, (8) allows us to express $\varepsilon_n^{(j)}$ in terms of the predictor errors introduced at the points $t_1, t_2, \ldots, t_n$. Thus, given the predictor errors, we can get insight into the effect of the parameters $j^*$, $m$ and $n$ on the iteration errors.

Let us introduce the $ms$-dimensional vector $U_n := e \otimes Y_n$, where $Y_n$ denotes the solution of (2). Then, we may define the *stage vector iteration errors*

$$\varepsilon_n := V_n - U_n, \quad \varepsilon_n^{(j)} := Y_n^{(j)} - Y_n, \quad j = j^*, \ldots, m + j^* - 1, \tag{9}$$

and the *predictor error* vector $\Delta_n$

$$\Delta_n := \begin{pmatrix} \delta_n \\ \delta_{n-1} \\ \vdots \\ \delta_1 \end{pmatrix}, \quad \delta_n := \theta P Y_{n-1} + (1-\theta) Y_n^{(1)} - Y_n, \quad \theta = 0, 1. \tag{10}$$

**Theorem 1** *Let the block rows of the matrix $S^k C$ be denoted by $[S^k C]^{(j)}$, $j = j^*, j^* + 1, \ldots, m + j^* - 1$. Then, for any vector of predictor errors $\Delta_n$, the iteration errors $\varepsilon_n^{(j)}$, $j = j^*, \ldots, m$, are given by*

$$\varepsilon_n^{(j)} = \Sigma_\theta^{(j)} \Delta_n, \quad \Sigma_\theta^{(j)} := ([C]^{(j)} \ [SC]^{(j)} \ [S^2 C]^{(j)} \ \ldots \ [S^{n-1} C]^{(j)}). \tag{11}$$

*If $\theta = 1$, then the error equation can be written as*

$$\begin{aligned} \varepsilon_n^{(j)} &= \widetilde{\Sigma}_1^{(j)} Y_0, \\ \widetilde{\Sigma}_1^{(j)} &:= \sum_{k=0}^{n-1} [S^k C]^{(j)} (P - (I - zA)^{-1} E)((I - zA)^{-1} E)^{n-k-1}. \end{aligned} \tag{12}$$

**Proof.** On substitution of $Y_n^{(j)} = Y_n + \varepsilon_n^{(j)}$ into (8), we obtain

$$\begin{aligned} \varepsilon_n^{(j)} &= [S]^{(j)} \varepsilon_{n-1} + [S]^{(j)} U_{n-1} + (1-\theta) Z^{j-1} Y_n^{(1)} - Y_n \\ &= [S]^{(j)} \varepsilon_{n-1} + [\theta Z^{j-1} P + \sum_{k=1}^{j-1} Z^{k-1} K] Y_{n-1} + (1-\theta) Z^{j-1} Y_n^{(1)} - Y_n \\ &= [S]^{(j)} \varepsilon_{n-1} + \theta Z^{j-1} P Y_{n-1} + (I - Z^{j-1})(I - Z)^{-1} K Y_{n-1} \\ &\qquad + (1-\theta) Z^{j-1} Y_n^{(1)} - Y_n \\ &= [S]^{(j)} \varepsilon_{n-1} + Z^{j-1} [\theta P Y_{n-1} + (1-\theta) Y_n^{(1)} - Y_n] \\ &= [S]^{(j)} \varepsilon_{n-1} + Z^{j-1} \delta_n, \end{aligned}$$

where $j = j^*, \ldots, m + j^* - 1$. Hence, $\varepsilon_n = S\varepsilon_{n-1} + C\delta_n$, and repeated application yields

$$\varepsilon_n = \sum_{k=0}^{n-1} S^k C \delta_{n-k}, \quad \varepsilon_n^{(j)} = \sum_{k=0}^{n-1} [S^k C]^{(j)} \delta_{n-k}, \quad j = j^*, \ldots, m + j^* - 1. \tag{13}$$

This leads to the representation (11). If $\theta = 1$, then it follows from (10) and (2) that

$$\delta_n := PY_{n-1} - Y_n = (P - (I - zA)^{-1}E)((I - zA)^{-1}E)^{n-1}Y_0, \tag{14}$$

and substitution into (13) yields the result (12). □

The $s \times s$ matrix $[S^k C]^{(j)}$ in (11) determines the amplification of the predictor error $\delta_{n-k}$ at the point $t_{n-k}$, and the accumulated amplification is determined by the matrix $\Sigma_\theta^{(j)}$. This amplification matrix, and therefore also $\varepsilon_n^{(j)}$, depends not only on $j$, $n$ and on the variable $z$, but also on the parameters $j^*$ and $m$. In general, $\varepsilon_n^{(j)}$ will decrease in magnitude as $j^*$ and $m$ increase. However, if for given $j$, $n$ and $j^*$, the value of $m$ becomes greater than $j + (n-1)(j^* - 1)$, then $\varepsilon_n^{(j)}$ does not depend on $m$ anymore. The result (12) takes the predictor formula into account, but again the matrix $[S^k C]^{(j)}$ plays a crucial role in the amplification matrix $\widetilde{\Sigma}_1^{(j)}$.

Let us assume that the predictor error vector $\Delta_n$ is bounded. Then, with respect to a norm $\|\cdot\|$ and for given values of $j^*$ and $n$, the region of convergence associated with (11) is defined by the set

$$\mathbf{C}_\theta(n, j^*) := \{z : \|\Sigma_\theta^{(j)}\| \to 0 \text{ as } m = j \to \infty\}. \tag{15}$$

Similarly, the region of convergence associated with (12) is defined by

$$\widetilde{\mathbf{C}}_1(n, j^*) := \{z : \|\widetilde{\Sigma}_1^{(j)}\| \to 0 \text{ as } m = j \to \infty\}. \tag{16}$$

Furthermore, adapting a definition given in ([13], p. 88), we define for (11) and (12) the averaged *speed (or rate) of convergence* by

$$R_\theta(n, j^*, j, m, z) := -\frac{1}{j} \log \|\Sigma_\theta^{(j)}\|, \quad \widetilde{R}_1(n, j^*, j, m, z) := -\frac{1}{j} \log \|\widetilde{\Sigma}_1^{(j)}\|. \tag{17}$$

## 3.2 A convergence theorem for $\theta = 0$

In order to determine the region of convergence and to get insight into the speed of convergence, we first derive an upper bound for $\|[S^k C]^{(j)}\|$. We confine our considerations to the case of arbitrary predictor formulas (i.e. $\theta = 0$). In deriving the upper bound for $\|[S^k C]^{(j)}\|$, an important tool is provided by the $\varepsilon$-pseudo-spectra of matrices which are defined as follows (see e.g. Reichel and Trefethen [11]):

**Definition 1** *Let $\|.\|_2$ denote the 2-norm and let $\varepsilon > 0$. Then, (i) $\mu$ is an $\varepsilon$-pseudo-eigenvalue of the matrix $M$ if $\|(\mu I - M)^{-1}\|_2 \geq \varepsilon^{-1}$, (ii) $\Lambda_\varepsilon(M)$ is the $\varepsilon$-pseudo-spectrum of $M$ if it contains all $\varepsilon$-pseudo-eigenvalues of $M$, and (iii) $\rho_\varepsilon(M)$ is the $\varepsilon$-pseudo-spectral radius of $M$ if it equals the maximal modulus of the points in $\Lambda_\varepsilon(M)$.*

Obviously, for any positive $\varepsilon$, all eigenvalues of $M$ are included in the $\varepsilon$-pseudo-spectrum of $M$. Furthermore, for a given matrix $M$ and parameter $\varepsilon$, and sufficiently large values of $|\mu|$, we have that $\|(\mu I - M)^{-1}\|_2 = |\mu^{-1}|(1 + \mathcal{O}(\mu^{-1})) < \varepsilon^{-1}$. Hence, the $\varepsilon$-pseudo-spectrum of $M$ constitutes a finite set in the complex plane.

**Theorem 2** *Let $\Lambda_\varepsilon(Z)$ and $\rho_\varepsilon(Z)$ respectively denote the $\varepsilon$-pseudo-spectrum and the $\varepsilon$-pseudo-spectral radius of $Z$, let $L_\varepsilon(Z)$ denote the length of the boundary $\partial\Lambda_\varepsilon(Z)$ of $\Lambda_\varepsilon(Z)$, and define*

$$\Gamma_\varepsilon(Z) := \frac{\varepsilon}{(\rho_\varepsilon(Z))^{j^*+1}} \max_{\partial\Lambda_\varepsilon(Z)} \|(\zeta I - Z)^{-1} Z^{j^*}\|_2, \quad \gamma_\varepsilon(Z) := \frac{1}{\varepsilon}\|K\|_2(\rho_\varepsilon(Z))^{j^*}.$$

*If $j \geq j^* + 1$, $m \to \infty$ and $\rho(Z) < 1$, then for any $\varepsilon > 0$ we have that*

$$\|[S^k C]^{(j)}\|_2 \leq \frac{L_\varepsilon(Z)}{2\pi\varepsilon}\Gamma_\varepsilon(Z)(\rho_\varepsilon(Z))^j(\gamma_\varepsilon(Z))^k, \tag{18}$$

$$\|\Sigma_0^{(j)}\|_2 \leq \frac{L_\varepsilon(Z)}{2\pi\varepsilon}\Gamma_\varepsilon(Z)(\rho_\varepsilon(Z))^j \frac{1-(\gamma_\varepsilon(Z))^n}{1-\gamma_\varepsilon(Z)}. \tag{19}$$

**Proof.** The result (18) can be proved by means of convolution properties of the Fourier transform. Because the proof is rather lengthy, it is given in the Appendix to this paper. The estimate (19) directly follows from (18) by writing

$$\|\Sigma_0^{(j)}\|_2 \leq \sum_{k=0}^{n-1} \|[S^k C]^{(j)}\|_2 \leq \frac{L_\varepsilon(Z)}{2\pi\varepsilon}\Gamma_\varepsilon(Z)(\rho_\varepsilon(Z))^j \sum_{k=0}^{n-1}(\gamma_\varepsilon(Z))^k.$$

□

The $\varepsilon$-pseudo-spectral radius $\rho_\varepsilon(Z)$ is continuous in $\varepsilon$ and monotonically decreasing to $\rho(Z)$ as $\varepsilon \to 0$. Since $L_\varepsilon(Z)/(2\pi\varepsilon)$ is bounded for all $\varepsilon$ and because there is always an $\varepsilon$ with $\rho_\varepsilon(Z) < 1$ (provided that $\rho(Z) < 1$), we conclude that, for fixed $n$, $\|\Sigma_0^{(j)}\|_2$ converges to 0 as $j$ increases. Thus, as a first corollary of Theorem 2 we have:

**Corollary 1** *If the conditions of Theorem 2 are satisfied, if $\theta = 0$ and $n$ is finite, then for all $j^*$ the convergence region of the PDIRKAS method is given by $\mathbf{C}_0(n, j^*) := \{z : \rho(Z(z)) < 1\}$.*

Table 1: Values of $\rho_\varepsilon(Z)$, $\Gamma_\varepsilon(Z)$ and $\gamma_\varepsilon(Z)$ for four-stage Radau IIA at $z = 10\mathrm{i}$.

| | $j^*$ | $\varepsilon = 1.00$ | $\varepsilon = 0.25$ | $\varepsilon = 0.1$ | $\varepsilon = 0.05$ | $\varepsilon = 0.001$ |
|---|---|---|---|---|---|---|
| $\rho_\varepsilon(Z)$ | | 2.478 | 1.451 | 1.080 | 0.903 | 0.542 |
| $\Gamma_\varepsilon(Z)$ | 2 | 0.177 | 0.503 | 0.823 | 1.063 | 1.910 |
| $\Gamma_\varepsilon(Z)$ | 3 | 0.053 | 0.268 | 0.558 | 0.807 | 1.899 |
| $\Gamma_\varepsilon(Z)$ | 4 | 0.014 | 0.114 | 0.305 | 0.516 | 1.852 |
| $\gamma_\varepsilon(Z)$ | 2 | 6.04 | 8.28 | 11.47 | 16.04 | 289.0 |
| $\gamma_\varepsilon(Z)$ | 3 | 14.97 | 12.02 | 12.39 | 14.49 | 156.60 |
| $\gamma_\varepsilon(Z)$ | 4 | 37.09 | 17.44 | 13.39 | 13.08 | 84.89 |

Furthermore, it follows from Theorem 2 that, given the values of $j^*$ and $n$, the speed of convergence is bounded below according to an inequality of the form

$$R_0(n, j^*, j, m, z) \geq -\log \rho_\varepsilon(Z(z)) - \frac{a_\varepsilon(n, j^*, z)}{j}, \tag{20}$$

where $a_\varepsilon(n, j^*, z)$ does not depend on $j$. This estimate illustrates the crucial role played by the $\varepsilon$-pseudo-spectral radius of the matrix $Z(z)$.

In order to see the effect of the quantities $\rho_\varepsilon(Z)$, $\Gamma_\varepsilon(Z)$ and $\gamma_\varepsilon(Z)$ on the convergence, we have listed their values in Table 1 for the the four-stage Radau IIA corrector with matrix $B = D$ as defined in [6] with $Z$ evaluated at the point $z = 10\mathrm{i}$ (this point is in the neighbourhood where experimentally the convergence speed is minimal). These figures together with the estimate (19) indicate that for larger values of $j$ and $n$, the convergence behaviour is largely determined by the factor $(\rho_\varepsilon(Z))^j(\gamma_\varepsilon(Z))^{n-1}$. Hence, given the value of $j^*$, roughly the same reduction factor is obtained if $jn^{-1}$ is constant.

## 3.3 Stiff and nonstiff convergence for $\theta = 0$

In this section, we consider the convergence in the neighbourhood of the origin (*nonstiff* convergence) and at infinity (*stiff* convergence). For the nonstiff convergence, it is convenient to have an alternative representation for the inequality (19). As a second corollary of Theorem 2 we have:

**Corollary 2** *Let the conditions of Theorem 2 be satisfied and define the matrix*

$$Z_0 := z^{-1}Z = (I - zB)^{-1}(A - B).$$

*Then (19) can be represented in the form*

$$\|\Sigma_0^{(j)}\|_2 \leq \frac{L_\varepsilon(Z_0)}{2\pi\varepsilon}|z|^{j-1}\Gamma_\varepsilon(Z_0)(\rho_\varepsilon(Z_0))^j \frac{1 - |z|^{(j^*-1)n}(\gamma_\varepsilon(Z_0))^n}{1 - |z|^{j^*-1}\gamma_\varepsilon(Z_0)}. \tag{21}$$

*Let $j$ and $j^*$ be fixed with $j^* > 1$. Then, the nonstiff convergence factor is uniformly bounded for all $n$.*

**Proof.** From the definition of the $\varepsilon$-pseudo-spectral radius it is easily seen that for any matrix $M$ and any constant $\alpha$, the relation $\rho_\varepsilon(\alpha M) = |\alpha|\rho_\delta(M)$, where $\delta = |\alpha|^{-1}\varepsilon$. Since $Z = zZ_0$, we have

$$\rho_\varepsilon(Z) = |z|\rho_\delta(Z_0), \quad \Gamma_\varepsilon(Z) = |z|^{-1}\Gamma_\delta(Z_0),$$

$$\gamma_\varepsilon(Z) = |z|^{j^*-1}\gamma_\delta(Z_0), \quad \frac{L_\varepsilon(Z)}{2\pi\varepsilon} = \frac{L_\delta(Z_0)}{2\pi\delta},$$

where $\delta := |z|^{-1}\varepsilon$. Hence,

$$\|[S^k C]^{(j)}\|_2 \leq \frac{L_\delta(Z_0)}{2\pi\delta}|z|^{j-1}\Gamma_\delta(Z_0)(\rho_\delta(Z_0))^j(|z|^{j^*-1}\gamma_\delta(Z_0))^k.$$

Because this inequality holds for any positive $\delta$, we may replace $\delta$ with $\varepsilon$ to obtain (21).

For $j^* > 1$, (21) implies

$$\|\Sigma_0^{(j)}\|_2 \leq \frac{L_\varepsilon(Z_0)}{2\pi\varepsilon}|z|^{j-1}\Gamma_\varepsilon(Z_0)(\rho_\varepsilon(Z_0))^j(1 + \mathcal{O}(z^{j^*-1})) \quad \text{as} \quad z \to 0.$$

Thus, this bound on the nonstiff convergence factor does not anymore depend on $n$. □

We remark that for $j^* = 1$, $\gamma_\varepsilon(Z_0) = \varepsilon^{-1}\|K\|_2\rho_\varepsilon(Z_0)$. Hence, unless we can find an $\varepsilon$ such that $\gamma_\varepsilon(Z_0) < 1$, the bound on $\|\Sigma_0^{(j)}\|_2$ will increase exponentially with $n$. Since $\|K\|_2 \to 1$ and $Z_0 \to A - B$ as $z \to 0$, we obtain the condition $\rho_\varepsilon(A - B) < \varepsilon$ which is usually not fulfilled.

For the stiff convergence we have:

**Corollary 3** *Let the conditions of Theorem 2 be satisfied, and let $j$ and $j^*$ be fixed. Then, the stiff convergence factor is uniformly bounded for all $n$.*

**Proof.** It follows from Theorem 2 and the observation $\|K(z)\|_2 = \mathcal{O}(z^{-1})$ as $z \to \infty$, that for all $j^*$

$$\|\Sigma_0^{(j)}\|_2 \leq \frac{L_\varepsilon(Z)}{2\pi\varepsilon}\Gamma_\varepsilon(Z(\infty))(\rho_\varepsilon(Z(\infty)))^j + \mathcal{O}(z^{-1}) \quad \text{as} \quad z \to \infty,$$

so that the stiff convergence factor is uniformly bounded in $n$. □

## 3.4 Minimal speed of convergence for $\theta = 0$

In order to see the effect of the value of $n$, $j^*$, $m$ and $j$ on the true speed of convergence as defined by (11) and (17), we have computed the minimal

Table 2: Minimal convergence speeds for $j = m = 32$ and $\theta = 0$.

| $n$ | $j^* = 1$ | $j^* = 2$ | $j^* = 3$ | $j^* = 4$ | $j^* = 32$ |
|---|---|---|---|---|---|
| 2 | 0.144 | 0.153 | 0.163 | 0.172 | 0.222 |
| 4 | 0.025 | 0.047 | 0.069 | 0.091 | 0.222 |
| 8 | −0.144 | −0.113 | −0.075 | −0.031 | 0.222 |

Table 3: Minimal convergence speeds for $n = 8$ and $\theta = 0$.

| $j = m$ | $j^* = 1$ | $j^* = 2$ | $j^* = 3$ | $j^* = 4$ | $j^* = m$ |
|---|---|---|---|---|---|
| 8 | −0.605 | −0.675 | −0.600 | −0.463 | 0.050 |
| 32 | 0.025 | 0.047 | 0.069 | 0.091 | 0.222 |
| 64 | 0.017 | 0.041 | 0.066 | 0.091 | 0.250 |

value of $R_0(n, j^*, j, m, z)$ in the lefthand $z$-plane. This value will be denoted by $R_0^*(n, j^*, j, m)$. Of course, $R_0^*(n, j^*, j, m)$ refers to a 'worst-case' situation, and restricting $z$ to special subregions (e.g. the negative axis) would lead to larger speeds of convergence. However, the qualitative behaviour would not be changed.

In particular, we consider the PC pair consisting of an unconditionally stable predictor and the four-stage Radau IIA corrector with matrix $B = D$ as in Table 1. Using the infinity norm, Table 2 lists $R_0^*(n, j^*, j, m)$ for a few values of $n$ and $j^*$ with $j = m = 32$. (We recall that for $j^* = m$, (4) reduces to the conventional iteration strategy without step parallelism.) These figures show the dramatic effect of $n$ on the amplification factors. It is also clear that the $n$-effect is less as $j^*$ is larger.

Table 4: Minimal convergence speeds and efficiency for $j = m, n = 8$, $N_{\text{seq}} = 96$ and $\theta = 0$.

| | $j^* = 1$ | $j^* = 2$ | $j^* = 3$ | ... | $j^* = 7$ |
|---|---|---|---|---|---|
| $j = m = 96 - 7j^*$ | 89 | 82 | 75 | ... | 47 |
| $R_0^*(n, j^*, j, m)$ | 0.075 | 0.082 | 0.091 | ... | 0.149 |
| $jR_0^*(n, j^*, j, m)$ | 6.7 | 6.7 | 6.8 | ... | 7.0 |

| | $j^* = 8$ | $j^* = 9$ | $j^* = 10$ | $j^* = 11$ | $j^* = 12$ |
|---|---|---|---|---|---|
| $j = m = 96 - 7j^*$ | 40 | 33 | 26 | 19 | 12 |
| $R_0^*(n, j^*, j, m)$ | 0.175 | 0.200 | 0.200 | 0.168 | 0.125 |
| $jR_0^*(n, j^*, j, m)$ | 7.0 | 6.6 | 5.2 | 3.2 | 1.5 |

Next, we computed $R_0^*(n, j^*, j, m)$ as a function of $m$ and $j^*$ for $n$ fixed with $j = m$. Table 3 lists results for $n = 8$. As expected, the performance improves as $m$ increases.

From a practical point of view, we have to take into account the sequential costs when discussing the performance of the iteration process. Recalling that the sequential costs of iteration across $n$ steps are measured by the value of $N_{\text{seq}} = (n-1)j^* + m$, we see that large values of $m$ are less alarming than they would be in conventional iteration processes with $j^* = m$, where the sequential costs after $n$ steps are given by $N_{\text{seq}} = nm$. As long as $j^*$ is less than the number of iterations required by conventional iteration, across-the-steps iteration will be more efficient. We illustrate this for the case where $n = 8$ and $N_{\text{seq}}$ is constant for all $j^*$. Table 4 lists values of $R_0^*(n, j^*, j, m)$ for $N_{\text{seq}} = 96$. After a rapid increase until $j^* = 9$, the convergence speed starts to decrease because $m$ becomes too small. We also listed the value of $jR_0^*(n, j^*, j, m)$ that may be considered as a measure of the efficiency of the iteration process after $j$ iterations. Surprisingly, for $j^* \leq 9$, the efficiency hardly depends on $j^*$. Apparently, the decrease of the number of iterations $m$ per step is fully compensated by the increase of $j^*$, until $m$ becomes too small at $j^* = 10$.

## 3.5 Minimal speed of convergence for $\theta = 1$

Finally, we study the effect of including the predictor formula into the convergence analysis (the given-predictor case with $\theta = 1$). Two special predictor formulas are considered, viz. the *modified correction* formula

$$\begin{aligned} Y_n^{(1)} - h_n(B \otimes I)F(Y_n^{(1)}) &= (E \otimes I)Y_{n-1}^{(j^*)} \\ &\quad + h_n((A-B) \otimes I)F((E^* \otimes I)Y_{n-1}^{(j^*)}), \end{aligned} \tag{22}$$

$$E^* = WX^{-1}, \qquad W := (\frac{1}{i}c^i), \qquad X := ((c-e)^{i-1}), \quad i = 1, \ldots, s,$$

and the *backward Euler* formula

$$Y_n^{(1)} - h(D^* \otimes I)F(Y_n^{(1)}) = (E \otimes I)Y_{n-1}^{(j^*)}, \quad D^* := \text{diag}(c). \tag{23}$$

For (22) and (23), the matrix $P$ occurring in the error formula (12) is defined by $P = (I - zB)^{-1}(E + z(A-B)E^*)$ and $P = (I - zD^*)^{-1}E$, respectively. Again using the infinity norm, Table 5 lists the minimal convergence speed $\widetilde{R}_1^*(n, j^*, j, m)$ in the lefthand $z$-plane. This table shows that, in spite of its low order, the backward Euler predictor is more effective than the high-order modified correction predictor. This indicates that the *stiff* iteration error components play a crucial role in the iteration process. Note that both cases show roughly the same increase of the convergence speed as $j^*$ increases.

Table 5: Minimal convergence speeds for $j = m = 32$ and $n = 4$.

| Predictor | Error formula | $j^* = 1$ | $j^* = 2$ | $j^* = 3$ | $j^* = 4$ | ... | $j^* = m$ |
|---|---|---|---|---|---|---|---|
| (22) | $\widetilde{R}_1^*(n, j^*, j, m, z)$ | 0.016 | 0.034 | 0.056 | 0.081 | ... | 0.212 |
| (23) | $\widetilde{R}_1^*(n, j^*, j, m, z)$ | 0.041 | 0.063 | 0.084 | 0.106 | ... | 0.238 |

## Acknowledgement

The authors are grateful to Prof. L. N. Trefethen for the interest he showed in our attempts to prove convergence for the PDIRKAS method. The use of $\varepsilon$-pseudo-spectral radius in the final proof of Theorem 2 was suggested by him when he visited CWI.

# Appendix: Proof of Theorem 2

The proof of Theorem 2 in this paper was given by the first author.

First, we show that in Theorem 2, $m \to \infty$ can be replaced by $m = \infty$, without sacrificing any generality. For $m = \infty$ the iteration process (7) is well defined. Furthermore, all the theory developed in the paper is still valid. Only the expression for the sequential costs should be reformulated. It is apparent from (7) that if $m = \infty$ and $\theta = 0$, $Y_n^{(j)}$ depends only on $Y_{n-1}^{(j^*+1)}, \ldots, Y_{n-1}^{(j^*-1+j)}$. Applying this repeatedly, we see that $Y_n^{(j)}$ depends only on $Y_1^{(j^*+1)}, \ldots, Y_1^{((n-1)(j^*-1)+j)}$. Therefore $Y_n^{(j)}$ is independent of $m$, provided that $m \geq (n-1)(j^*-1)+j$. Hence, $[S^k C]^{(j)}$ with $m = \infty$ equals $[S^k C]^{(j)}$ with $m \geq (k-1)(j^*-1)+j$.

The proof consists of two parts. First, $[S^k C]^{(j)}$ will be written as a line integral along the unit circle (see Lemma 1). Thereafter, we obtain an upper bound for this integral (see Lemma 2). In the proofs of these lemmas we shall use matrix Fourier analysis and more specifically the convolution property and the inverse Fourier transform.

**Lemma 1** *If $\theta = 0$, $m = \infty$ and $\rho(Z) < 1$ then, for $j \geq j^* + 1$*

$$[S^k C]^{(j)} = \frac{1}{2\pi \mathrm{i}} \oint_{|\zeta|=1} [R(Z,\zeta)K]^k \, R(Z,\zeta) \, \zeta^{(k-1)j^*+j-1} \mathrm{d}\zeta \; Z^{j^*},$$

*with $R(Z,\zeta) = (\zeta I - Z)^{-1}$.*

**Proof.** In the case $\theta = 0$ the iterates $Y_n^{(j^*+1)}, Y_n^{(j^*+2)}, \ldots$, do not depend on $Y_{n-1}^{(j^*)}$, see equation (7). Therefore we redefine the block vectors $V_n$ and $C$ and

block matrix $S$ as follows

$$V_n = \begin{pmatrix} Y_n^{(j^*+1)} \\ Y_n^{(j^*+2)} \\ \vdots \end{pmatrix}, \quad C = \begin{pmatrix} Z^{j^*} \\ Z^{j^*+1} \\ \vdots \end{pmatrix}, \tag{24}$$

and

$$S = \begin{pmatrix} Z^{j^*-1}K & Z^{j^*-2}K & \dots & K & 0 & \dots & \\ Z^{j^*}K & Z^{j^*-1}K & \dots & ZK & K & 0 & \dots \\ Z^{j^*+1}K & Z^{j^*}K & \dots & Z^2K & ZK & K & 0 & \dots \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \end{pmatrix}.$$

With these changes $V_n = SV_{n-1} + CY_n^{(1)}$ still holds. It can easily be seen that also equation (11) is still valid. The new matrix $S$ is a so-called Toeplitz matrix. This is a matrix that represents a convolution. For showing that $S$ is a convolution operator and for doing Fourier analysis we shall use as general argument for the operator $S$ any $V$ of the form

$$V = \begin{pmatrix} V^{(j^*+1)} \\ V^{(j^*+2)} \\ \vdots \end{pmatrix},$$

where $V^{(j)}$, $j = j^*+1, j^*+2, \dots$, are matrices of order $s$. For simplifying the Fourier analysis, we introduce the notation: $V(j) = V^{(j^*+1+j)}$, $j = 0, 1, \dots$. In particular, this notation will be applied to $C, SC, S^2C, \dots$. Furthermore, these block vectors are considered as being sequences of $s \times s$ matrices, that is with the block vector $V$ corresponds the sequence $\{V(j)\}_{j=0}^{\infty}$.

The new matrix $S$ is of the form

$$\begin{pmatrix} A_0 & A_1 & A_2 & \dots & & \\ A_{-1} & A_0 & A_1 & A_2 & \dots & \\ A_{-2} & A_{-1} & A_0 & A_1 & A_2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix}.$$

In our case $A_j = Z^{j^*-1-j}K$ if $j \leq j^* - 1$ and $A_j = 0_{s*s}$ if $j > j^* - 1$. Though $S$ is a double infinite matrix, $S^kV$ is well defined for any sequence $V$ of $s \times s$ matrices. This is because for any $k$ and $r$, the $r^{th}$ row of $S^k$ contains a finite number of nonzero entries. In terms of the sequence $\{A_j\}_{j=-\infty}^{\infty}$ the product $SV$ can be written as

$$[SV](l) = \sum_{j=0}^{\infty} A_{-l+j}V(j) = \sum_{j=0}^{\infty} A_{-(l-j)}V(j), \quad l = 0, 1, \dots . \tag{25}$$

Equation (25) shows that $S$ is a convolution operator: $SV = \{A_{-l}\} * V$. To obtain a formula for $[S^k C]^{(j)}$ we shall employ Fourier transforms along with their convolution property. The Fourier transform of an infinite sequence of matrices $\{A_l\}_{l=-\infty}^{\infty}$ is defined by

$$\mathcal{F}(\{A_l\})(\omega) = \sum_{l=-\infty}^{\infty} A_l \mathrm{e}^{-\mathrm{i}\omega l},$$

and the convolution property gives

$$\mathcal{F}(SV) = \mathcal{F}(\{A_{-l}\} * V) = \mathcal{F}(\{A_{-l}\})\, \mathcal{F}(V). \tag{26}$$

Let $H(\omega)$ be defined by

$$\begin{aligned} H(\omega) &= \mathcal{F}(\{A_{-l}\})(\omega) = \sum_{p=-\infty}^{\infty} A_{-p} \mathrm{e}^{-\mathrm{i}\omega p} \\ &= \sum_{p=-\infty}^{\infty} A_p \mathrm{e}^{\mathrm{i}\omega p} = \sum_{p=-\infty}^{j^*-1} Z^{j^*-1-p} K \mathrm{e}^{\mathrm{i}\omega p} \\ &= \sum_{p=0}^{\infty} Z^p K \mathrm{e}^{-\mathrm{i}\omega p} \mathrm{e}^{\mathrm{i}\omega(j^*-1)} = (I - \mathrm{e}^{-\mathrm{i}\omega} Z)^{-1} K \mathrm{e}^{\mathrm{i}\omega(j^*-1)}. \end{aligned}$$

The series $\sum_{p=0}^{\infty} Z^p \mathrm{e}^{-\mathrm{i}\omega p}$ is convergent because $\rho(\mathrm{e}^{-\mathrm{i}\omega} Z) = \rho(Z) < 1$. Using the convolution property (26) repeatedly, we can calculate $\mathcal{F}(S^k C)$ as follows

$$\begin{aligned} \mathcal{F}(S^k C)(\omega) &= \mathcal{F}(\{A_{-l}\} * \ldots * \{A_{-l}\} * C) \\ &= H^k(\omega) \mathcal{F}(C)(\omega). \end{aligned}$$

For the block vector $C$ (see (24)), we have that $C(j) = Z^{j^*+j}, j = 0, 1 \ldots$. The Fourier transform of $C$ is

$$\mathcal{F}(C)(\omega) = \sum_{p=0}^{\infty} Z^p \mathrm{e}^{-\mathrm{i}\omega p} Z^{j^*} = Z^{j^*} (I - \mathrm{e}^{-\mathrm{i}\omega} Z)^{-1}.$$

Finally the Fourier transform of $S^k C$ is

$$\mathcal{F}(S^k C)(\omega) = [(I - \mathrm{e}^{-\mathrm{i}\omega} Z)^{-1} K]^k (I - \mathrm{e}^{-\mathrm{i}\omega} Z)^{-1} Z^{j^*} \mathrm{e}^{\mathrm{i}\omega k(j^*-1)}.$$

Since we have been able to obtain an explicit expression for the Fourier transform of $S^k C$, the inverse Fourier transform can be used to calculate $[S^k C]^{(j)}$ as follows

$$[S^k C]^{(j)} = [S^k C](j - j^* - 1) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathcal{F}(S^k C)(\omega)\, \mathrm{e}^{\mathrm{i}\omega(j-j^*-1)} \mathrm{d}\omega$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi}[(I-\mathrm{e}^{-\mathrm{i}\omega}Z)^{-1}K]^k(I-\mathrm{e}^{-\mathrm{i}\omega}Z)^{-1}\mathrm{e}^{\mathrm{i}\omega[k(j^*-1)+j-j^*-1]}\mathrm{d}\omega\ Z^{j^*}. \quad (27)$$

In this formula and in all formulas below, it is assumed that $j \geq j^* + 1$. Substituting $\zeta = \mathrm{e}^{\mathrm{i}\omega}$, (27) can be written as

$$\begin{aligned}[S^kC]^{(j)} &= \frac{1}{2\pi\mathrm{i}}\oint_{|\zeta|=1}[(I-\zeta^{-1}Z)^{-1}K]^k(I-\zeta^{-1}Z)^{-1}\zeta^{(k-1)j^*+j-k-2}\mathrm{d}\zeta\ Z^{j^*} \\ &= \frac{1}{2\pi\mathrm{i}}\oint_{|\zeta|=1}[(\zeta I-Z)^{-1}K]^k(\zeta I-Z)^{-1}\zeta^{(k-1)j^*+j-1}\mathrm{d}\zeta\ Z^{j^*}, \quad (28)\end{aligned}$$

which proves Lemma 1. □

**Remark.** The integral in (28) can be calculated by using the calculus of residues. This gives

$$[S^kC]^{(j)} = \sum_{i_1+\ldots+i_{k+1}=(k-1)j^*+j-k-1} Z^{i_1}K\ldots Z^{i_k}KZ^{i_{k+1}+j^*}, \quad (29)$$

where the indices $i_1,\ldots,i_{k+1}$ assume all positive values as long as

$$i_1+\ldots+i_{k+1} = (k-1)j^*+j-k-1$$

is satisfied. It is difficult to find a sharp upper bound based on (29).

A suitable bound based on the previous lemma can be obtained by using the concept of the $\varepsilon$-pseudo-spectrum of a matrix, which was defined in Section 3.2. We shall prove:

**Lemma 2** *Let $\Lambda_\varepsilon(Z)$, $\rho_\varepsilon(Z)$,$\Gamma_\varepsilon(Z)$ and $\gamma_\varepsilon(Z)$ be defined as in Theorem 2. If $\theta = 0$, $m = \infty$ and $\rho(Z) < 1$, then for any $j \geq j^* + 1$ and $\varepsilon > 0$*

$$\| [S^kC]^{(j)} \|_2 \leq \frac{L_\varepsilon(Z)}{2\pi\varepsilon}\,\Gamma_\varepsilon(Z)\,(\rho_\varepsilon(Z))^j\,[\gamma_\varepsilon(Z)]^k.$$

**Proof.** The integrand of the integral in (28) is analytic outside $\Lambda_\varepsilon(Z)$ for any $\varepsilon > 0$. Therefore

$$[S^kC]^{(j)} = \frac{1}{2\pi\mathrm{i}}\oint_{\partial\Lambda_\varepsilon(Z)}[(\zeta I-Z)^{-1}K]^k(\zeta I-Z)^{-1}\zeta^{(k-1)j^*+j-1}\mathrm{d}\zeta\ Z^{j^*},$$

with $\varepsilon$ any positive real number. Now the integral can be bounded in the following way

$$\begin{aligned}\| [S^kC]^{(j)} \|_2 &\leq \frac{1}{2\pi}\| K \|_2^k \oint_{\partial\Lambda_\varepsilon(Z)} \| (\zeta I-Z)^{-1}Z^{j^*} \|_2 \| (\zeta I-Z)^{-1} \|_2^k \\ &\qquad \cdot|\zeta|^{(k-1)j^*+j-1}|\mathrm{d}\zeta| \\ &\leq \frac{L_\varepsilon(Z)}{2\pi}\| K \|_2^k\,(\frac{1}{\varepsilon})^{k+1}\Gamma_\varepsilon(Z)[\rho_\varepsilon(Z)]^{kj^*+j}.\end{aligned}$$

This proves Lemma 2. □

The assertion (18) in Theorem 2 is identical with the assertion of Lemma 2.

# References

[1] A. Bellen. *Parallelism across the steps for difference and differential equations*, pages 22–35. Lecture Notes in Mathematics 1386. Springer-Verlag, 1987.

[2] A. Bellen, R. Vermiglio, and M. Zennaro. Parallel ODE-solvers with step-size control. *JCAM*, 31:277–293, 1990.

[3] J. C. Butcher. *The numerical analysis of ordinary differential equations, Runge–Kutta and general linear methods.* Wiley, New York, 1987.

[4] P. Chartier. *Parallelism in the numerical solutions of initial value problems for ODEs and DAEs.* PhD thesis, Université de Rennes I, France, 1993.

[5] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems.* Springer-Verlag, 1991.

[6] 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.

[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] P. J. van der Houwen, B. P. Sommeijer, and W. A. van der Veen. Parallel iteration across the steps of high order Runge–Kutta methods for nonstiff initial value problems. *J. Comp. Appl. Math.*, 60:309–329, 1995.

[9] K. R. Jackson and S. P. Nørsett. The potential for parallelism in Runge–Kutta methods, Part I: RK formulas in standard form. *SIAM J. Numer. Anal.*, 32:49–82, 1995.

[10] I. Lie. Some aspects of parallel Runge–Kutta methods. Technical Report 3/87, Dept. of Mathematics, University of Trondheim, 1987.

[11] L. Reichel and L. N. Trefethen. Eigenvalues and pseudo-eigenvalues of Toeplitz matrices. *Linear Algebra Appl.*, 162/164:153–185, 1992.

[12] W. A. van der Veen. Step-parallel algorithms for stiff initial value problems. *Computers & mathematics with applications*, 30(11):9–23, 1995.

[13] D. M. Young. *Iterative solution of large linear systems.* Academic Press, New York, 1971.

# Chapter 5

# Step-parallel Algorithms for Stiff Initial Value Problems

W.A. van der Veen

**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 [5, 6]. 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 [5], 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 the present 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.

*CR Subject Classification (1991):* G1.7
*Keywords & Phrases :* numerical analysis, Runge-Kutta methods, parallelism

*Note:* The research reported in this paper was supported by the Technology Foundation (STW) in The Netherlands.

# 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 save to advance to the next point on the $t$-axis, and at the same time provides an initial guess (the advancing strategy).

Step-parallel methods go back to Miranker and Liniger [9] 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 [2, 1] which is based on Steffensen iteration (see also Chartier [3]) .

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 [5]. 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 [5], 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 [11] 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

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

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 non-scalar 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

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

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,

$$\begin{aligned} Y_{n+1}^0 &\quad \text{to be defined by the predictor formula,} && (2)\\ Y_{n+1}^j &= Y_{n+1}^{j-1} - (I - h_{n+1}DJ_n)^{-1}(Y_{n+1}^{j-1} - EY_n - h_{n+1}AF(Y_{n+1}^{j-1})), && (3)\\ Y_{n+1} &= Y_{n+1}^m. \end{aligned}$$

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 [5, 6], 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 [5], 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

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

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

$$\Delta(y_{n+1}^j, y_{n+1}^{j-1}) < Tol_{corr}, \qquad y_{n+1}^j = Y_{n+1,s}^j, \tag{5}$$

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 $Y^j_{n+1,i}$, $i = 1, \ldots, s$, 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 [5, 6] that their application to the test equation $y' = \lambda y$ (using fixed steps) gives an iteration process that is convergent for every $\lambda$ in the left half plane. Therefore, these PDIRK methods and the corresponding Radau IIA correctors have the same accuracy and stability properties, provided that $Tol_{corr}$ 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 [5] 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:

$$\begin{aligned} Y_{n+1}^0 \quad & \text{to be defined by the predictor formula,} \\ Y_{n+1}^j \;=\; & Y_{n+1}^{j-1} - (I - h_{n+1}DJ_n)^{-1} \\ & \cdot(Y_{n+1}^{j-1} - EY_n^{j^*(n)+j-1} - h_{n+1}AF(Y_{n+1}^{j-1})). \end{aligned} \tag{6}$$

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 [11], 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 [6]. 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., $Y_{n+1}^0 =$ Extrapolation$(Y_n^{j^*(n)})$. 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:

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

and

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

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

$$\begin{aligned} Y_n^{j^*(n)+j} &= Y_n^{j^*(n)+j-1} - (I - h_n DJ_{n-1})^{-1} \\ &\quad \cdot (Y_n^{j^*(n)+j-1} - EY_{n-1}^{j^*(n-1)+j^*(n)+j-1} - h_n AF(Y_n^{j^*(n)+j-1})), \end{aligned}$$

$$\begin{aligned} Y_{n+1}^{j} &= Y_{n+1}^{j-1} - (I - h_{n+1} DJ_n)^{-1} \\ &\quad \cdot (Y_{n+1}^{j-1} - EY_n^{j^*(n)+j-1} - h_{n+1} AF(Y_{n+1}^{j-1})), \end{aligned}$$

are calculated concurrently until the iterate in $I_n$ satisfies (5). Note, that both calculations use $Y_n^{j^*(n)+j-1}$. Also, the iterates $Y_n^{j^*(n)+j}$ and $Y_{n-1}^{j^*(n-1)+j^*(n)+j}$ 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 [11]. 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 the present 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 $Y_{n+1}^j$,$Y_n^{j^*(n)+j}$, $Y_{n-1}^{j^*(n-1)+j^*(n)+j}$, ... , 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

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

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

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

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

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

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

$$res(B, n, j) = \Delta(e_s^T B, e_s^T Y_{n-1}^{j+j^*(n-1)} + h_n e_s^T A F(B)).$$

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 well 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 $\lambda$ lying in the left half plane [7, 11]. 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

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

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

$$res(G_n^j, n, j) = \Delta(e_s^T G_n^j, e_s^T Y_{n-1}^{j+j^*(n-1)} + h_n e_s^T A F(G_n^j)).$$

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

$$res(Y_n^j, n, j) < \gamma\ \min(p_{rel}\ res(G_n^j, n, j), p_{abs}\ Tol), \tag{7}$$

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 $res(Y_n^j, n, j)$ 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 $\min(10^{-5}, 10^{-3} Tol)$.

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

$$\Delta(y_1^j, y_1^{j-1}) < Tol_1$$

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 $I_n$:

- $j^*(n) \leq maxj^*$,
- $res(Y_n^j, n, j) < reslim \quad \text{for} \quad j > jconv$,
- $\Delta(y_n^j, y_n^{j-1}) < 1 \quad \text{for} \quad j \geq 2$.

Here $j$ assumes all values for which $Y_n^j$ is not sufficiently accurate. Furthermore, $maxj^*$, $reslim$ and $jconv$ are method parameters (see Section 6 for their values). In view of the possible initial growth, the integer valued parameter $jconv$ 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

$$err_n = \begin{cases} \Delta(y_n^{j^*(n)}, e_s^T G_n^{j^*(n)}) & \text{if} \quad n > 1 \\ \Delta(y_1^{j^*(1)}, y_1^1) & \text{if} \quad n = 1, \end{cases}$$

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

$$h_{n+1} = h_n / \max(0.6, \min(3.0, \frac{1}{0.8} \sqrt[s]{\frac{err_n}{Tol}})). \tag{8}$$

Conversely, if $err_n \geq Tol$ then the step is rejected and (8) 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 $p_{abs} = p_{rel} = 0.5$.

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 (7) 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 (7). For this verification we need $F(Y_n^{j^*(n)})$ and $F(G_n^{j^*(n)})$. After these function evaluations, we advance to the interval $I_{n+1}$ and compute $Y_{n+1}^0$. Hence, $F(Y_n^{j^*(n)})$ is calculated before $F(Y_{n+1}^0)$ can be calculated. Because $F(Y_n^{j^*(n)})$ 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 $F(Y_{n+1}^0)$ simultaneously with $F(Y_n^j)$. Only when $Y_n^j$ satisfies condition (7), we really advance the iteration process to interval $I_{n+1}$, otherwise we just ignore $F(Y_{n+1}^0)$ 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, $F(Y_n^{j^*(n)})$ and $F(Y_{n+1}^0) = F(E_{n+1} Y_n^{j^*(n)})$ 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 $F(Y_{n+1}^0) = F(E_{n+1}Y_n^j)$ and $F(G_n^j)$ simultaneously with $F(Y_n^j)$.

# 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 $\min(10^{-5}, 10^{-3}\,Tol)$. 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 $err_n < Tol$. In that case the initial guess for the steplength is given by

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

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 $res(Y_{n-k}^0, n-k, 0)$, i.e.

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

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:

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

and

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

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

$$\Delta(y_n^j, y_n^{j-1}) < Tol_1,$$

with $Tol_1 = 10^{-4}$. 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 $Tol$, $Tol_{corr}$, $K$ and $h_0$. $Tol$ is the upper bound for the local error estimate, and $Tol_{corr}$ determines when the iteration process in the successive intervals can be terminated (cf (5)). Since a relatively small value of $Tol_{corr}$ only slightly increases the number of intervals treated simultaneously, while the effective costs remain approximately the same, we have chosen $Tol_{corr} = 10^{-12}$, 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 $maxj^*$, $jconv$, and $res-lim$, 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 [10] 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 [4], 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:

$$\begin{aligned} \frac{dy_1}{dt} &= -0.04\, y_1 + 10^4 y_2 y_3 \\ \frac{dy_2}{dt} &= 0.04\, y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2 \\ \frac{dy_3}{dt} &= 3 \cdot 10^7 y_2^2 \\ y(0) &= (1,0,0)^T \end{aligned} \tag{9}$$

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

We also include two van der Pol equations:

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= 50(1 - y_1^2)y_2 - y_1 \\ y(0) &= (2,0)^T \end{aligned} \tag{10}$$

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

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= ((1 - y_1^2)y_2 - y_1)10^6 \\ y(0) &= (2, -0.66)^T \end{aligned} \tag{11}$$

on [0,2] as the fourth. These ODEs have changes in their components in an almost discontinuous way (especially (11)).

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

$$\begin{aligned} \frac{dy_1}{dt} &= -\frac{1}{\varepsilon}(y_1 - cos(y_2)) - sin(y_2) \\ \frac{dy_2}{dt} &= 1 \\ y(0) &= (1,0)^T \end{aligned} \tag{12}$$

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 [8]:

$$\begin{aligned} \frac{dy_i}{dt} &= \frac{5 - y_i}{RC} - \frac{K}{C} g(y_{i-1}, y_i), \qquad i = 1, \ldots, 4, \\ R &= 5000, \qquad C = 0.2 \cdot 10^{-12}, \qquad K = 2 \cdot 10^{-4}, \\ g(u,v) &= (\max(u-1,0))^2 - (\max(u-v,0))^2, \\ y(0) &= (5, 0.5, 5, 0.5)^T, \end{aligned} \tag{13}$$

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

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

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

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 $Tol_{corr} = 10^{-9}$. In Table 1 (see appendix), the influence of $K$ is shown for the first test problem. For this small value of $Tol_{corr}$, 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 the 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 $Tol_{corr} = 10^{-9}$ instead of $Tol_{corr} = 10^{-12}$. As can be seen from these tables, PDIRKAS(EXT,4) is almost as good as PDIRKAS(EXT,10). If the parameter $Tol_{corr}$ 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 (10)-(11), 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) & 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.

## Acknowledgements

The author wishes to thank Prof. dr. P.J. van der Houwen and Dr. B.P. Sommeijer for their help during the preparation of this paper.

## References

[1] A. Bellen. *Parallelism across the steps for difference and differential equations.* in: Lecture Notes in Mathematics 1386. Springer-Verlag, 1987.

[2] A. Bellen, R. Vermiglio, and M. Zennaro. *Parallel ODE-solvers with step-size control. JCAM 31*, pages 277–293, 1990.

[3] P. Chartier. *Parallelism in the numerical solutions of initial value problems for ODEs and DAEs.* Thesis. Université de Rennes I, France, 1993.

[4] E. Hairer, C. Lubich, and M. Roche. *The Numerical solution of differential-algebraic Systems by Runge-Kutta Methods.* Lecture Notes in Mathematics 1409. Springer-Verlag, 1989.

[5] P.J. van der Houwen and B.P. Sommeijer. *Iterated Runge-Kutta methods on parallel computers. SIAM J. Sci. Stat. Comput. 12*, pages 1000–1028, 1991.

[6] P.J. van der Houwen and B.P. Sommeijer. *Analysis of parallel diagonally implicit iteration of Runge-Kutta methods. APNUM 11*, pages 169–188, 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*, pages 293–312, 1994.

[8] W. Kampowski, P. Rentrop, and W. Schmidt. *Classification and numerical simulation of electric circuits.* Math. Inst. Tech. Univ. Munchen, 1991.

[9] W.L. Miranker and W. Liniger. *Parallel methods for the numerical integration of ordinary differential equations. Math. Comp. 21*, pages 303–320, 1967.

[10] B.P. Sommeijer. *Parallel-iterated Runge-Kutta methods for stiff ordinary differential equations. JCAM 45*, pages 151–168, 1993.

[11] 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. To appear in APNUM*, 1995.

# Appendix

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

| $K$ | $Tol$ | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| 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 | $Tol$ | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 $Tol_{corr} = 10^{-9}$

Table 3: Results for the Robertson kinetics example (9)

| Method | *Tol* | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 (10)

| Method | *Tol* | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 (11)

| Method | *Tol* | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 (12)

| Method | *Tol* | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 (13)

| Method | *Tol* | $N$ | $nsd$ | $K_{max}(K_{av})$ | $C_{eff}$ | $j^*_{av}$ | $N_{reject}$ | $m_{av}$ |
|---|---|---|---|---|---|---|---|---|
| (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 |

# Chapter 6

# Parallel Iterative Linear Solvers for Multistep Runge–Kutta Methods

E. Messina[1], J. J. B. de Swart and W. A. van der Veen

**Abstract**

This paper deals with solving stiff systems of differential equations by implicit Multistep Runge–Kutta (MRK) methods. For this type of methods, nonlinear systems of dimension sd arise, where s is the number of Runge–Kutta stages and d the dimension of the problem. Applying a Newton process leads to linear systems of the same dimension, which can be very expensive to solve in practice. Like in [13], where the one-step RK methods were considered, we approximate these linear systems by s systems of dimension d, which can be solved in parallel on a computer with s processors. In terms of Jacobian evaluations and LU-decompositions, the k-step s-stage MRK applied with PILSMRK on s processors is equally expensive as the widely used k-step Backward Differentiation Formula on 1 processor, whereas the stability properties are better than that of BDF. If both methods perform the same number of Newton iterations, then the accuracy delivered by the new method is also higher than that of BDF.

*AMS Subject Classification (1991):* Primary: 65L05. Secondary: 65F05, 65F50.

*CR Subject Classification (1991):* G.1.7, G.4

*keywords & Phrases:* numerical analysis, Newton iteration, Multistep Runge–Kutta methods, parallelism.

*Note:* The research reported in this paper was partly supported by the Technology Foundation (STW) in the Netherlands.

[1]Dipartimento di Matematica e Applicazioni "R. Caccippoli", University of Naples "Federico II", Via Cintia, I-80126 Naples, Italy,

# 1 Introduction

For solving the stiff initial value problem (IVP)

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \quad y, f \in \mathbf{R}^d, \quad t_0 \leq t \leq t_e, \tag{1}$$

a widely used class of methods is that of the Backward Differentiation Formulae (BDFs)

$$y_n = (\kappa^T \otimes I) y^{(n-1)} + h_n \beta f(y_n).$$

Here, $\otimes$ denotes the Kronecker product and the vector $y^{(n-1)}$ is defined by $(y_{n-k}^T, \ldots, y_{n-1}^T)^T$, where $y_j^T$ approximates the solution at $t = t_j$ and $k$ is the number of previous steppoints that are used for the computation of the approximation in the current time interval. The stepsize $t_{n+1} - t_n$ is denoted by $h_n$. The scalar $\beta$ and the $k$-dimensional vector $\kappa$ contain the method parameters. They depend on $h^{(n)}$, which is the vector with $k$ previous stepsizes defined by $h^{(n)} := (h_{n-k+1}, \ldots, h_n)^T$. In the sequel, $I$ stands for the identity matrix and $e_i$ for unit vector in the $i$th direction. The dimensions of $I$ and $e_i$ may vary, but will always be clear from the context.

For example, the popular codes DASSL [18] and VODE [2] are based on BDFs. However, a drawback of BDFs is the loss of stability if the number of steppoints $k$ increases. As a consequence of Dahlquist's order barrier, no $A$-stable BDF can exceed order 2. Moreover, BDFs are not zero-stable for $k > 6$.

A promising class of methods that can overcome these drawbacks of BDFs are the Multistep Runge–Kutta (MRK) methods, which are of the form

$$y_n = (\chi^T \otimes I) y^{(n-1)} + h_n (\alpha^T \otimes I) F(Y_n), \tag{2}$$

where $Y_n$ is the solution of the equation

$$R(Y_n) = 0, \quad R(Y_n) := Y_n - (G \otimes I) y^{(n-1)} - h_n (A \otimes I) F(Y_n). \tag{3}$$

Here, $Y_n$ is the so-called stage vector of dimension $sd$, whose components $Y_{ni}$ represent approximations to the solution at $t = t_{n-1} + c_i h_n$, where $c := (c_1, \ldots, c_s)^T$ is the vector of abscissae and $s$ is the number of Runge–Kutta stages. The vector $F(Y_n)$ contains the derivative values $f(Y_{ni})$. The arrays $\alpha$, $\chi$, $A$ and $G$ contain method parameters and are of dimension $s \times 1$, $k \times 1$, $s \times s$ and $s \times k$, respectively. These parameters and the abscissae $c_i$ depend on $h^{(n)}$. We remark that a way of circumventing this dependence on $h^{(n)}$ is interpolating the previous steppoints, so that they are equally spaced. However, this strategy adds local errors and does not allow good stepsize flexibility, see [19, p.68].

Stability has been investigated for fixed stepsizes in the literature. Even for large values of $k$, these methods have "surprisingly" good stability properties [9, p.296]. For example, MRKs of Radau type with $s = 3$ remain stiffly stable for $k \leq 28$ and have modest error constants [19, p.13].

A drawback of using MRKs is the high cost of solving the non-linear system (3) of dimension $sd$ every time step. Normally, one uses a (modified) Newton process to solve this non-linear system. This leads to a sequence of iterates $Y_n^{(0)}, Y_n^{(1)}, \ldots, Y_n^{(m)}$ which are obtained as solutions of the $sd$-dimensional linear systems

$$(I - A \otimes h_n J_n)(Y_n^{(j)} - Y_n^{(j-1)}) = -R(Y_n^{(j-1)}), \quad j = 1, 2, \ldots, m, \qquad (4)$$

where $J_n$ is the Jacobian of the function $f$ in (1) evaluated in $t_n$, the starting vector $Y_n^{(0)}$ is defined by some predictor formula, and $Y_n^{(m)}$ is accepted as approximation to $Y_n$. If we use Gaussian elimination to solve these linear systems, then this would cost $\frac{2}{3}s^3d^3$ arithmetic operations for the $LU$-decompositions.

In order to reduce these costs, one can bring the Newton matrix $I - A \otimes h_n J_n$ to block diagonal form by means of similarity transformations [3] resulting in

$$\begin{aligned} (I - T^{-1}AT \otimes h_n J_n)(X_n^{(j)} - X_n^{(j-1)}) &= -(T^{-1} \otimes I)R(Y_n^{(j-1)}), \qquad (5) \\ Y_n^{(j)} &= (T \otimes I)X_n^{(j)}, \qquad j = 1, 2, \ldots, m. \end{aligned}$$

Here, $T^{-1}AT$ is of (real) block diagonal form. Every block of $T^{-1}AT$ corresponds with an eigenvalue pair of $A$. If the eigenvalue of $A$ is complex, then the block size of the associated block in $T^{-1}AT$ is 2, if the eigenvalue is real, then the block size is 1. The $LU$-costs are now reduced to $\frac{2}{3}d^3$ and $\frac{16}{3}d^3$ for the blocks of size 1 and 2, respectively. Hairer & Wanner used this approach in their code RADAU5 [10]. The blocks of the linear system (5) are now decoupled, so that the use of $\sigma$ processors reduces the effective costs to $\frac{16}{3}d^3$, where $\sigma$ is the number of blocks in $T^{-1}AT$ . Notice that pairs of stage values can be computed concurrently, i.e. it is possible to do function evaluations, transformations and vector updates for pairs of stages in parallel if $\sigma$ processors are available.

By exploiting the special structure of the $2d$-dimensional linear systems in (5), it is possible to reduce the costs of solving these systems (see e.g. [1]). Let $\xi_j \pm i\eta_j$ be an eigenvalue pair and assume that the matrix of the corresponding linear system is of the form

$$\begin{pmatrix} I - \xi_j h_n J_n & -\eta_j h_n J_n \\ \eta_j h_n J_n & I - \xi_j h_n J_n \end{pmatrix}. \qquad (6)$$

One easily checks that the inverse of (6) is

$$(I \otimes \Gamma^{-1}) \begin{pmatrix} I - \xi_j h_n J_n & \eta_j h_n J_n \\ -\eta_j h_n J_n & I - \xi_j h_n J_n \end{pmatrix}, \qquad (7)$$

$$\Gamma = I - 2\xi_j h_n J_n + (\xi_j^2 + \eta_j^2) h_n^2 J_n^2.$$

Using $\sigma$ processors, the $\mathcal{O}(d^3)$ costs of this approach are $\frac{8}{3}d^3$ ($2d^3$ for the computation of $J_n^2$ and $\frac{2}{3}d^3$ for the $LU$-decomposition of $\Gamma$). On $\sigma$ processors, an MRK using this implementation strategy is 4 times more expensive in terms of $\mathcal{O}(d^3)$ costs than a BDF, for which we only have to solve linear systems with a matrix of the form $I - h_n \beta J_n$.

In this paper we reduce the implementational costs of MRKs to a further extent by following the approach of [13]. Here, the matrix $A$ is approximated by a matrix $B$ with positive distinct eigenvalues and the iterates $Y_n^{(j)}$ in (4) are computed by means of the inner iteration process

$$\begin{aligned}
&(I - B \otimes h_n J_n)(Y_n^{(j,\nu)} - Y_n^{(j,\nu-1)}) = -(I - A \otimes h_n J_n) Y_n^{(j,\nu-1)} + C_n^{(j-1)}, \\
&C_n^{(j-1)} := (I - A \otimes h_n J_n) Y_n^{(j-1)} - R(Y_n^{(j-1)}).
\end{aligned} \tag{8}$$

The index $\nu$ runs from 1 to $r$ and $Y_n^{(j,r)}$ is accepted as the solution $Y_n^{(j)}$ of the Newton process (4). Furthermore, $Y_n^{(j,0)} = Y_n^{(j-1)}$. Since the matrix $B$ in (8) has distinct eigenvalues, applying a similarity transformation $Q$ that diagonalizes $B$, i.e. $BQ = QD$ where $D$ is a diagonal matrix, leads to:

$$\begin{aligned}
(I - D \otimes h_n J_n)(X_n^{(j,\nu)} - X_n^{(j,\nu-1)}) = &-(I - Q^{-1} A Q \otimes h_n J_n) X_n^{(j,\nu-1)} \\
&+ (Q^{-1} \otimes I) C_n^{(j-1)}, \quad \nu = 1, \ldots, r.
\end{aligned} \tag{9}$$

The system (9) consists of $s$ decoupled systems of dimension $d$ which can be solved in parallel. Every processor computes a stage value. The costs for the $LU$-decompositions are now reduced to $\frac{2}{3}d^3$ on $s$ processors. Notice that in order to ensure the non-singularity of the matrix $(I - D \otimes h_n J_n)$ the positiveness of the eigenvalues of $B$ is required. In analogy with [13] we will refer to (8) as PILSMRK, Parallel Linear System solver for Multistep Runge–Kutta methods. The combination of modified Newton and PILSMRK will be called the Newton-PILSMRK method.

We will discuss several strategies to choose $B$ such that the inner iterates in (8) converge quickly to the Newton iterates in (4). Experiments show that, if we apply more than 2 Newton iterations, then only 1 inner iteration suffices to find the Newton iterate. This means that in terms of $LU$-decompositions and Jacobian evaluations a $k$-step, $s$-stage Newton-PILSMRK on $s$ processors is as expensive as a $k$-step BDF on 1 processor, whereas the stability properties of Newton-PILSMRK are better. If both methods perform the same number of function evaluations, then the accuracies delivered by Newton-PILSMRK are also higher than that of BDF. It turns out that the convergence behaviour of the inner iteration process becomes better if $k$ increases. In particular, the inner iteration process for MRKs converges faster than that for the one-step RK methods proposed in [13].

The outline of the paper is as follows. § 2 briefly describes how to determine the MRK parameters. In § 3 we investigate the convergence of the inner iteration process for several choices of the matrix $B$, and we consider the stability of the overall method in § 4. Numerical experiments in § 5 show the performance of the proposed methods on a number of test problems. Finally, we draw some conclusions in § 6.

## 2 Construction of MRKs

A large class of multistep Runge-Kutta methods consists of multistep collocation methods, which were first investigated by Guillou and Soulé [7]. Later, Lie and Nørsett [15] considered the MRKs of Gauss type and Hairer and Wanner [9] those of Radau type. In the useful thesis of Schneider [19] on MRKs for stiff ODEs and DAEs a lot of properties of MRKs and further references can be found.

For convenience of the reader we briefly describe here how one can compute $c$, $G$ and $A$. Alternative ways of deriving these parameters can be found in [9] and [19]. In a multistep collocation method, the solution is approximated by a so-called collocation polynomial. Given $y^{(n)}$, $h^{(n)}$ and $c$, we define the collocation polynomial $u(t)$ of degree $s+k-1$ by

$$\begin{aligned} u(t_j) &= y_j, & j &= n-k+1,\ldots,n, \\ u'(t_n+c_ih_n) &= f(u(t_n+c_ih_n)), & i &= 1,\ldots,s. \end{aligned}$$

The stage vector $Y_n$ is then given by $(u(t_n+c_1h_n)^T,\ldots,u(t_n+c_sh_n)^T)^T$. In order to compute $u(t)$, we expand it in terms of polynomials $\phi_i$ and $\psi_i$ of degree $s+k-1$, given by

$$\begin{aligned} \phi_i(\tau_j) &= \delta_{ij}, & j &= 1,\ldots,k, & i &= 1,\ldots,k, \\ \phi_i'(c_j) &= 0, & j &= 1,\ldots,s, & i &= 1,\ldots,k, \\ \psi_i(\tau_j) &= 0, & j &= 1,\ldots,k, & i &= 1,\ldots,s, \\ \psi_i'(c_j) &= \delta_{ij}, & j &= 1,\ldots,s, & i &= 1,\ldots,s. \end{aligned}$$

Here, $\delta_{ij}$ denotes the Kronecker tensor, $\tau$ is the dimensionless coordinate $\frac{t-t_n}{h_n}$ and $\tau_j = \frac{t_{n-k+j}-t_n}{h_n}$, $j=1,\ldots,k$. In terms of these polynomials the expansion of $u(t)$ is given by

$$\begin{aligned} u(t_n+\tau h) &= \sum_{j=1}^{k}\phi_j(\tau)y_{n-k+j} + h_n\sum_{i=1}^{s}\psi_i(\tau)u'(t_n+c_ih_n) \\ &= \sum_{j=1}^{k}\phi_j(\tau)y_{n-k+j} + h_n\sum_{i=1}^{s}\psi_i(\tau)f(u(t_n+c_ih_n)). \end{aligned}$$

Clearly, the MRK parameters read $G_{ij} = \phi_j(c_i)$, $A_{ij} = \psi_j(c_i)$, $\alpha_j = \phi_j(1)$ and $\chi = \psi_j(1)$. Notice that the order of the approximations $u(t_n + c_i h_n)$, the so-called *stage order* of the MRK, is $s + k - 1$.

To construct the polynomials $\phi_i(\tau)$ and $\psi_i(\tau)$, we expand them as

$$\phi_i(\tau) = \sum_{m=0}^{s+k-1} d^{\phi}_{m,i} \tau^m \qquad \text{and} \qquad \psi_i(\tau) = \sum_{m=0}^{s+k-1} d^{\psi}_{m,i} \tau^m.$$

Substituting the first expression into the defining conditions yields

$$\begin{pmatrix} 1 & \tau_1 & \tau_1^2 & \tau_1^3 & \dots & \tau_1^{s+k-1} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & \tau_k & \tau_k^2 & \tau_k^3 & \dots & \tau_k^{s+k-1} \\ 0 & 1 & 2c_1 & 3c_1^2 & \dots & (s+k-1)c_1^{s+k-2} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 1 & 2c_s & 3c_s^2 & \dots & (s+k-1)c_s^{s+k-2} \end{pmatrix} \begin{pmatrix} d^{\phi}_{0,i} \\ \vdots \\ d^{\phi}_{s+k-1,i} \end{pmatrix} = e_i. \qquad (10)$$

The matrix of order $s + k$ in (10) will be denoted by $W$. For the polynomials $\psi_i(\tau)$ we derive analogously

$$W \begin{pmatrix} d^{\psi}_{0,i} \\ \vdots \\ d^{\psi}_{s+k-1,i} \end{pmatrix} = e_{k+i}.$$

To compute the $A$ and the $G$, we evaluate $\phi_i(\tau)$ and $\psi_i(\tau)$ in $\tau = c_j$ for $j = 1, \dots, s$, yielding

$$\begin{aligned} \phi_i(c_j) &= (1\ c_j\ \dots c_j^{s+k-1})\, W^{-1} e_i, \\ \psi_i(c_j) &= (1\ c_j\ \dots c_j^{s+k-1})\, W^{-1} e_{i+k}. \end{aligned}$$

Introducing

$$V = \begin{pmatrix} 1 & c_1 & \dots & c_1^{s+k-1} \\ \vdots & \vdots & & \vdots \\ 1 & c_s & \dots & c_s^{s+k-1} \end{pmatrix},$$

the matrices $G$ and $A$ are respectively given by

$$G = VW^{-1}(e_1, \dots, e_k) \qquad \text{and} \qquad A = VW^{-1}(e_{k+1}, \dots, e_{k+s}).$$

We now construct the abscissae vector $c$ such that we have superconvergence in the steppoints. Only stiffly accurate Multistep Runge–Kutta methods will be considered, i.e. $c_s = 1$. This means that we can omit steppoint formula (2) and obtain $y_{n+1}$ from $y_{n+1} = (e_s^T \otimes I) Y_n$. A well known subclass of stiffly

accurate MRK methods are the multistep Radau methods, which are $A(\alpha)$-stable. Their set of collocation points $c_1, \ldots, c_{s-1}$ is given (see [9, p.294]) as the roots in the interval [0,1] of

$$\sum_{j=1}^{k} \frac{1}{c_i - \tau_j} + \sum_{\substack{j=1 \\ j \neq i}}^{s} \frac{2}{c_i - c_j} = 0, \quad i = 1, \ldots, s-1.$$

We call the order of approximation $y_{n+1}$ to $y(t_{n+1})$ the *steppoint order* or, more loosely, the *order* of the MRK. This choice of $c$ leads to steppoint order $2s + k - 2$.

The appendix to this paper lists the MRK parameters for $s \in \{2, 4\}$ and $k \in \{2, 3\}$.

# 3 Convergence of the inner iteration process

We now discuss the choice of the matrix $B$ in (8) such that the inner iteration process converges rapidly. If we define the *inner iteration error* by $\epsilon_n^{(j,\nu)} := Y_n^{(j,\nu)} - Y_n^{(j)}$, then (4) and (8) yield the recursion

$$\epsilon_n^{(j,\nu)} = Z(h_n J_n)\epsilon_n^{(j,\nu-1)}, \qquad Z(h_n J_n) := (I - B \otimes h_n J_n)^{-1}((A - B) \otimes h_n J_n).$$

Applying the method to Dahlquist's test equation

$$y' = \lambda y, \quad \lambda \in \mathbf{C}, \tag{11}$$

this recursion reduces to

$$\epsilon_n^{(j,\nu)} = Z(z_n)\epsilon_n^{(j,\nu-1)}, \qquad z_n := h_n \lambda. \tag{12}$$

Let $\mu(\cdot)$ be the logarithmic norm associated with the Euclidean norm, which can be expressed as $\mu(S) := \frac{1}{2}\lambda_{\max}(S + S^T)$, where $\lambda_{\max}(\cdot)$ denotes the algebraically largest eigenvalue of a matrix (see e.g. [8, p.61]). For dissipative problems $\mu(J_n) \leq 0$. The following lemma states that the inner iteration process converges for dissipative problems at least as fast as for the 'most unfavourable' linear test equation. For the proof of this lemma we refer to [17].

**Lemma 1** *If* $\mu(J_n) \leq 0$, *then* $\|Z(h_n J_n)^\nu\|_2 \leq \max\{\|Z^\nu(z_n)\|_2 : Re(z_n) \leq 0\}$.

In § 3.1 and § 3.2 we treat two choices for the matrix $B$ that make $Z(z_n)$ 'small' in some sense. To measure $Z(z_n)$ we use the following quantities:

- $\rho^{(j)}(z_n)$, the (averaged) rate of convergence after $j$ iterations in $z_n$, defined by

$$\rho^{(j)}(z_n) := \sqrt[j]{\|Z(z_n)^j\|_2}.$$

- $\rho_\infty^{(j)}$, the stiff convergence rate after $j$ iterations, defined by

$$\rho_\infty^{(j)} := \sqrt[j]{\|Z_\infty^j\|_2}, \quad Z_\infty := \lim_{z_n \to \infty} Z(z_n) = (I - B^{-1}A).$$

  $Z_\infty$ will be referred to as the *stiff amplification matrix.*

- $\rho^{(j)}$, the maximal convergence rate after $j$ iterations, defined by

$$\rho^{(j)} := \max_{\mathrm{Re}(z_n) \leq 0} \{\rho^{(j)}(z_n)\}.$$

Notice that because of the maximum principle and the fact that $\rho^{(j)}(z_n)$ is symmetric with respect to the real axis, $\rho^{(j)}(z_n)$ takes its maximum at the positive imaginary axis:

$$\rho^{(j)} := \max_{(x_n) \geq 0} \{\rho^{(j)}(\mathrm{i}x_n)\}.$$

Since $A$ depends on $h^{(n)}$, $B$ also depends on $h^{(n)}$. Consequently, the procedure for constructing $B$ has to be carried out every time $h^{(n)}$ changes and should not be too expensive.

## 3.1 Constructing $B$: Crout decomposition

Let $L$ be the lower triangular matrix of the Crout decomposition of $A$, i.e. $L$ is lower triangular such that $L^{-1}A$ is upper triangular with ones on the diagonal. As proposed in [14], we choose $B = L$. The stiff amplification matrix takes the form $I - L^{-1}A$, which is strictly upper triangular. Consequently, $\rho_\infty^{(j)} = 0$ for $j \geq s$. For reasons that will become clear in § 3.2, we will refer to this inner iteration process as PILSMRK($L$,$I$).

Table 1 lists the values of $\rho^{(j)}$ for a few PILSMRK($L$,$I$) methods for the case with constant stepsizes. As a reference we included the one-step Radau IIA methods. From this table we see that, for the worst-case situation, the convergence of the MRKs is better than that of the one-step Runge–Kutta methods.

In practice, the rate of convergence in other points of the complex plane is also of interest. Figure 1 shows $\rho^{(j)}(z_n)$ along the imaginary axis $z_n = \mathrm{i}x_n$, $x_n \in \mathbf{R}$ for PILSMRK($L$,$I$) with $(k = 3, s = 4)$ method with constant stepsizes for $j = 1, 2, 3, 4$ and $j = \infty$. From this figure we clearly see that $\rho_\infty^{(j)} = 0$ for $j \geq s$.

In order to see the effect of variable stepsizes on the convergence rate, we define

$$\omega_i = h_i / h_{i-1} \quad \text{for} \quad i = n - k + 2, \ldots, n$$

Table 1: Values of $\rho^{(j)}$ for several PILSMRK($L$,$I$) methods with constant stepsizes.

| $s$ | $k$ | Order | $j=1$ | $j=2$ | $j=3$ | $j=4$ | ... | $j=\infty$ |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 0.24 | 0.21 | 0.20 | 0.19 | ... | 0.18 |
| | 2 | 4 | 0.19 | 0.17 | 0.16 | 0.16 | ... | 0.15 |
| | 3 | 5 | 0.17 | 0.15 | 0.15 | 0.14 | ... | 0.14 |
| 4 | 1 | 7 | 0.59 | 0.54 | 0.53 | 0.52 | ... | 0.51 |
| | 2 | 8 | 0.54 | 0.50 | 0.49 | 0.48 | ... | 0.47 |
| | 3 | 9 | 0.52 | 0.48 | 0.47 | 0.46 | ... | 0.44 |
| 8 | 1 | 15 | 1.03 | 0.94 | 0.91 | 0.90 | ... | 0.86 |
| | 2 | 16 | 0.98 | 0.92 | 0.89 | 0.88 | ... | 0.84 |
| | 3 | 17 | 0.97 | 0.92 | 0.89 | 0.87 | ... | 0.82 |

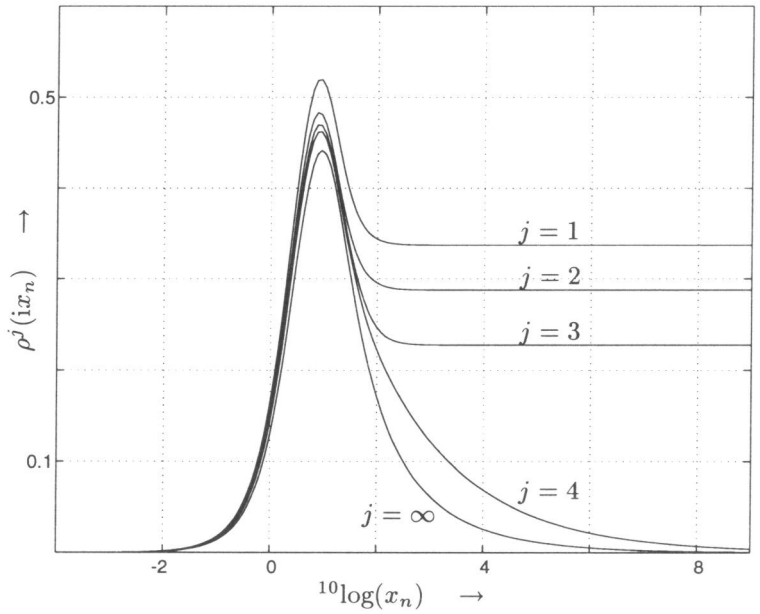


Figure 1: $\rho^{(j)}(ix_n)$ for PILSMRK($L$,$I$) with $k=3, s=4$.

and plotted $\rho^{(j)}$ as function of $\omega_i$ for several PILSMRK methods. Here, $\omega_i \in [0.2, 2]$, since in an actual implementation, a reasonable factor by which subsequent stepsizes are multiplicated lies in this interval. These plots revealed that the influence of variable stepsizes on the rate of convergence is modest. E.g., for $k=2$, $s=4$, $\rho^{(j)} \in [0.45, 0.58]$, $\forall j$, and for $k=3$, $s=4$, $\rho^{(j)} \in [0.495, 0.525]$, $\forall j$.

## 3.2 Constructing $B$: Schur-Crout decomposition

Before approximating the matrix $A$ by the lower factor of the Crout decomposition, we first transform $A$ to 'a more triangular form', the real Schur form. In the usual eigenvalue problem the eigenvalues and eigenvectors are unknown. However, the problem with which we are faced here is computing a real Schur form, given the eigenvalues and eigenvectors of $A$. Below we specify precisely how we do this. We remark that this construction is not developed to be cheap, but such that we are able to exploit the freedom in the real Schur form.

Let $\gamma$ be the vector with eigenvalues of $A$, and $\xi$ and $\eta$ be the real and imaginary part of $\gamma$, respectively, i.e. $\gamma = \xi + \mathrm{i}\eta$. Order the components of $\gamma$ as follows (we will motivate this choice later):

$$|\eta_i^2/\xi_i| \geq |\eta_j^2/\xi_j| \quad \text{for} \quad i > j. \tag{13}$$

In addition, if $|\eta_i^2/\xi_i| = |\eta_{i+1}^2/\xi_{i+1}|$, then $\eta_i > 0$. This sequence is such that real eigenvalues have the lowest index in $\gamma$ and complex eigenvalues are ordered in conjugated pairs by increasing value of $\eta_j^2/\xi_j$, while the eigenvalue with positive imaginary part comes first within a pair. The matrices $A$ treated in this paper have at most one real eigenvalue, so that we do not have to sort real eigenvalues.

Let $e_j^{\mathrm{r}} + \mathrm{i}e_j^{\mathrm{i}}$ be the eigenvector belonging to $\gamma_j$, such that $\|e_j^{\mathrm{r}} + \mathrm{i}e_j^{\mathrm{i}}\|_2 = 1$ and $e_{j1}^{\mathrm{i}} = 0$. For all matrices $A$ that are of interest here, this scaling turns out to exist. Define

$$E = ( \, e_1^{\mathrm{r}} \quad e_1^{\mathrm{i}} \quad e_3^{\mathrm{r}} \quad e_3^{\mathrm{i}} \quad \ldots \quad e_{s-1}^{\mathrm{r}} \quad e_{s-1}^{\mathrm{i}} \, )$$

if $A$ has only complex eigenvalues, and

$$E = ( \, e^{\mathrm{r}} \quad e_2^{\mathrm{r}} \quad e_2^{\mathrm{i}} \quad e_4^{\mathrm{r}} \quad e_4^{\mathrm{i}} \quad \ldots \quad e_{s-1}^{\mathrm{r}} \quad e_{s-1}^{\mathrm{i}} \, )$$

if $A$ has one real eigenvalue with eigenvector $e^{\mathrm{r}}$. One easily verifies that the matrix $E^{-1}AE$ is block diagonal with $2 \times 2$ blocks

$$\begin{pmatrix} \xi_j & \eta_j \\ -\eta_j & \xi_j \end{pmatrix},$$

and one block equal to $\xi_1$ if $\eta_1 = 0$. We orthonormalize the columns of $E$ by a Gram-Schmidt process, i.e. we construct a lower block triangular matrix $K$ such that $EK$ is orthogonal. This matrix $EK$ transforms $A$ to a matrix $H$:

$$H := (EK)^{-1}A(EK) = K^{-1}(E^{-1}AE)K. \tag{14}$$

Since $K$ is lower triangular and $E^{-1}AE$ is block diagonal, it is clear that $H$ is lower block triangular. Notice that $H$ is a real Schur form of $A$.

We now rotate the diagonal blocks of $H$ by means of a matrix $\Theta$ such that $\Theta^{-1}H\Theta$ is 'more suitable' to be approximated by its lower Crout factor. Define

$$\Theta = \mathrm{diag}(\Theta_j), \qquad \Theta_j = \begin{pmatrix} \cos\theta_j & \sin\theta_j \\ -\sin\theta_j & \cos\theta_j \end{pmatrix}, \qquad \Theta_1 = 1 \quad \text{if} \quad \eta_1 = 0,$$

and $S = \Theta^{-1}H\Theta$. Here, $j \in \{2, 4, \ldots, s-1\}$ if $\eta_1 = 0$ and $j \in \{1, 3, \ldots, s-1\}$ if $\eta_1 \neq 0$. The lower factor of the Crout decomposition of $S$ is again denoted by $L$. Remark that the stiff amplification matrix $I - L^{-1}S$ is block diagonal with $2 \times 2$ blocks containing only one non-zero entry. One easily verifies that this entry is given by

$$-S_{j,j+1}/S_{j,j}, \tag{15}$$

where

$$\begin{aligned} S_{j,j} &= \frac{1}{2}((H_{j,j} - H_{j+1,j+1})\cos(2\theta_j) - (H_{j,j+1} + H_{j+1,j})\sin(2\theta_j) \\ &\quad + (H_{j,j} + H_{j+1,j+1})), \\ S_{j,j+1} &= \frac{1}{2}((H_{j+1,j} + H_{j,j+1})\cos(2\theta_j) + (H_{j,j} - H_{j+1,j+1})\sin(2\theta_j) \\ &\quad + (H_{j,j+1} + H_{j+1,j})), \end{aligned}$$

and the diagonal blocks of $H$ and $S$ are of the form

$$\begin{pmatrix} H_{j,j} & H_{j,j+1} \\ H_{j+1,j} & H_{j+1,j+1} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} S_{j,j} & S_{j,j+1} \\ S_{j+1,j} & S_{j+1,j+1} \end{pmatrix}.$$

We choose $\theta_j$ such that the absolute value of (15) is minimized. By using Maple [4] we established that this is done for $\theta_j$ given by

$$\arctan \frac{H_{j,j}H_{j+1,j} + H_{j,j+1}H_{j+1,j+1} + \sqrt{\det(H)(\|H\|_F^2 - 2\det(H))}}{H_{j+1,j}^2 + H_{j+1,j+1}^2 - \det(H)} \bmod \pi,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. The matrix $B$ with real eigenvalues that approximates $A$ is thus given by $B = ULU^T$, where $U := EK\Theta$ is an orthogonal matrix. Applying a similarity transformation $Q$ such that $BQ = QD$, we again arrive at scheme (9). The linear system solver resulting from this Schur-Crout approach will be referred to as PILSMRK($L$,$U$), where the $U$ indicates that we have transformed $A$ before approximating it by $L$.

We now illustrate the idea that moved us to sort the eigenvalues as in (13). For simplicity of notation, we assume here that $s = 4$. If the first order expansion of $Z(z_n)$ for small $z_n$ is given by

$$Z(z_n) := z_n Z_0 + \mathcal{O}(z_n^2),$$

Table 2: Values of $\rho^{(j)}$ for several PILSMRK($L$,$U$) methods with constant stepsizes.

| $s$ | $k$ | Order | $j=1$ | $j=2$ | $j=3$ | $j=4$ | ... | $j=\infty$ |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 0.24 | 0.21 | 0.20 | 0.19 | ... | 0.18 |
| | 2 | 4 | 0.18 | 0.16 | 0.16 | 0.15 | ... | 0.15 |
| | 3 | 5 | 0.15 | 0.14 | 0.13 | 0.13 | ... | 0.13 |
| 4 | 1 | 7 | 0.55 | 0.49 | 0.47 | 0.47 | ... | 0.44 |
| | 2 | 8 | 0.50 | 0.45 | 0.43 | 0.43 | ... | 0.41 |
| | 3 | 9 | 0.47 | 0.42 | 0.41 | 0.40 | ... | 0.39 |
| 8 | 1 | 15 | 0.91 | 0.78 | 0.74 | 0.72 | ... | 0.65 |
| | 2 | 16 | 0.88 | 0.76 | 0.72 | 0.70 | ... | 0.62 |
| | 3 | 17 | 0.86 | 0.74 | 0.70 | 0.68 | ... | 0.61 |

then $Z_0 = A - B$. It can be verified that for the Schur-Crout approach $Z_0$ is of the form

$$Z_0 = U \begin{pmatrix} 0 & \zeta_{12} & 0 & 0 \\ 0 & \zeta_{22} & 0 & 0 \\ 0 & \zeta_{32} & 0 & \zeta_{34} \\ 0 & \zeta_{42} & 0 & \zeta_{44} \end{pmatrix} U^T,$$

where

$$\begin{pmatrix} \zeta_{32} \\ \zeta_{42} \end{pmatrix} = v \begin{pmatrix} S_{31} \\ S_{41} \end{pmatrix}, \quad v = -\frac{1}{S_{21}} \frac{\eta_1^2}{\xi_1}.$$

In order to keep the lower triangular part of $Z_0$ as small as possible, the best we can do is sorting the eigenvalues such that those with the smallest value of $\eta_k^2/\xi_k$ come first.

Table 2 and Figure 2 are the analogues of Table 1 and Figure 1 for PILSMRK-($L$,$U$). The worst-case $\rho^{(j)}$-values in Table 2 are smaller than those in Table 1. The difference between PILSMRK($L$,$I$) and PILSMRK($L$,$U$) becomes larger in favour of PILSMRK($L$,$U$) as $s$ increases. This can be understood by realizing that for the Crout option, we approximate the matrix $A$ with $s^2$ parameters by a matrix $B$ with $s(s+1)/2$ entries, whereas for the Schur-Crout case, the matrix $U^TAU$ with $s(s+1)/2+l$ nonzero entries, where $l$ is the number of complex conjugated eigenvalue pairs, is approximated by $U^TBU$ with $s(s+1)/2$ parameters. In addition, the advantage of PILSMRK($L$,$U$) over PILSMRK($L$,$I$) is that the stiff convergence rate $\rho_\infty^{(j)}$ vanishes for $j > 1$, which is confirmed by Figure 2. The extra price that we have to pay is the construction of the real Schur decomposition of $A$ every time $\omega_j$ changes for some $j$. Since in practice $s \ll d$, we do not consider this as a serious drawback.

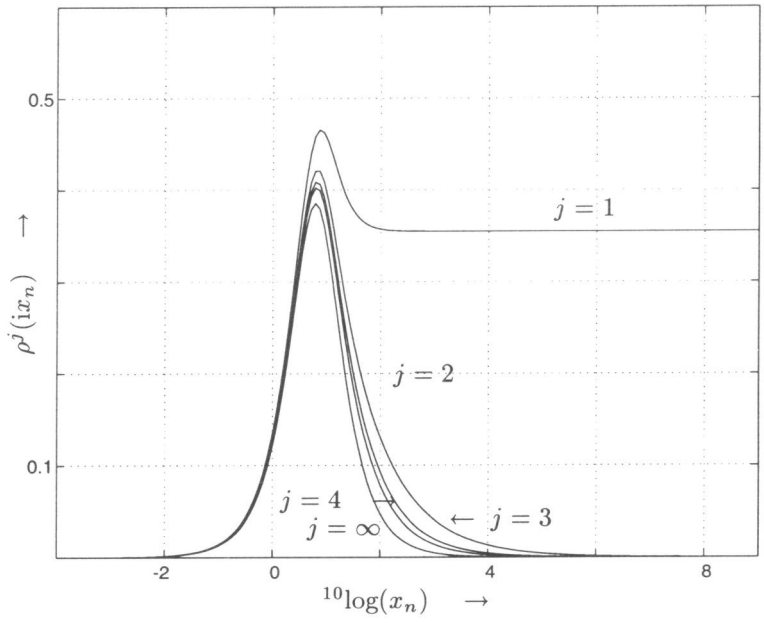


Figure 2: $\rho^{(j)}(ix_n)$ for PILSMRK(*L*,*U*) with $k = 3, s = 4$.

**Remark 1**

There is freedom in the choice of the transformation matrix $Q$ that diagonalizes $B$. If $X$ is a matrix with eigenvectors of $B$ and $\Sigma$ and $P$ are diagonal and permutation matrices, respectively, then for every matrix $Q$ of the form

$$Q = X\Sigma P, \tag{16}$$

we have that $BQ = QD$. Starting with a fixed matrix $X$, we determined $\Sigma$ and $P$ in (16) such that the elements of $Q$ and $Q^{-1}$ are not too large. □

**Remark 2**

Another approach for finding a suitable matrix $B$, based on rotations that minimize $\rho^{(1)}$, can be found in [12]. □

The matrices $D$ and $Q$ that result from the Crout and Schur-Crout approaches are given in the appendix to this paper for several values of $k$ and $s$.

# 4 Stability

In this paragraph we investigate the stability of the corrector formula (3) and the PILSMRK method (8) for test equation (11) solved with constant stepsizes $h$. We only consider stiffly accurate methods, i.e. $y_n = e_s^T Y_n^{(m,r)}$.

Following [19] we write (3) in the form

$$y^{(n)} = M(z)y^{(n-1)}, \qquad M(z) = \begin{pmatrix} N \\ e_s^T(I - zA)^{-1}G \end{pmatrix}, \qquad z := h\lambda,$$

where the $(k-1) \times k$ matrix $N$ is given by

$$N = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & 0 \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}.$$

The *stability region* is defined by

$$\mathcal{S} := \{z \in \mathbf{C} \mid \rho(M(z)) < 1\}, \tag{17}$$

where $\rho(\cdot)$ denotes the spectral radius function. We use the quantity $\widehat{D}^{(mr)}$ to measure the stability region (see [9, p.268]), where

$$\widehat{D} := -\inf\{\mathrm{Re}(z) \mid z \notin \mathcal{S}\}.$$

In practice, the PILSMRK method will be used to solve the corrector only approximately. Therefore we do not attain the stability of the corrector. For conducting a stability analysis for the PILSMRK methods we assume that in each step $m$ outer and $r$ inner iterations are carried out. In addition we assume that the predictor is only based on the stage vector in the previous steppoint,

$$Y_n^{(0,r)} = (P \otimes I)Y_{n-1}^{(m,r)}, \tag{18}$$

where $P$ is an $s \times s$ matrix. From (12) and (3) we derive a recursion in $\nu$:

$$Y_n^{(j,\nu)} = Z(z)Y_n^{(j,\nu-1)} + (I - zB)^{-1}Gy^{(n-1)}.$$

An elementary manipulation, in which we use $Y_n^{(j,0)} = Y_n^{(j-1,r)}$, leads to a recursion in $j$:

$$Y_n^{(j,r)} = Z^r(z)Y_n^{(j-1,r)} + (I - Z^r(z))(I - zA)^{-1}Gy^{(n-1)}.$$

Substituting (18) yields the following recursion in time:

$$Y_n^{(m,r)} = Z^{mr}(z)PY_{n-1}^{(m,r)} + (I - Z^{mr}(z))(I - zA)^{-1}Gy^{(n-1)}, \tag{19}$$

which we write in the form

$$\begin{pmatrix} y^{(n)} \\ Y_n^{(m,r)} \end{pmatrix} = M^{(mr)}(z) \begin{pmatrix} y^{(n-1)} \\ Y_{n-1}^{(m,r)} \end{pmatrix},$$

$$M^{(mr)}(z) = \begin{pmatrix} M_{11}^{(mr)}(z) & M_{12}^{(mr)}(z) \\ M_{21}^{(mr)}(z) & M_{22}^{(mr)}(z) \end{pmatrix}.$$

From (19) we see that

$$M_{21}^{(mr)}(z) = (I - Z^{mr}(z))(I - zA)^{-1}G, \qquad M_{22}^{(mr)}(z) = Z^{mr}(z)P.$$

Since we restrict ourselves here to stiffly accurate methods,

$$M_{11}^{(mr)} = \begin{pmatrix} N \\ e_s^T M_{21}^{(mr)} \end{pmatrix}, \qquad M_{12}^{(mr)} = \begin{pmatrix} O_{k-1,s} \\ e_s^T M_{22}^{(mr)} \end{pmatrix},$$

where $O_{ij}$ denotes an $i \times j$ zero matrix. Notice that this linear stability analysis does not distinguish between outer and inner iterations. In analogy with (17) we define the *stability region after mr iterations* by

$$\mathcal{S}^{(mr)} := \{z \in \mathbf{C} \mid \rho(M^{(mr)}(z)) < 1\}$$

and the stability measure

$$\widehat{D}^{(mr)} := -\inf\{\mathrm{Re}(z) \mid z \notin \mathcal{S}^{(mr)}\}.$$

It is clear that

$$\lim_{mr \to \infty} \widehat{D}^{(mr)} = \widehat{D}.$$

Table 3 and 4 list $\widehat{D}^{(mr)}$-values for the $k$-step $s$-stage MRK of Radau type for $k \in \{1,2,3,4\}$ and $s \in \{2,4,8\}$ with PILSMRK($L$,$I$) and PILSMRK($L$,$U$), respectively. For $s \leq 4$, we used the predictor that extrapolates the previous stage values, i.e. we determined $P$ in (18) such that $Y_n^{(0,r)}$ has maximal order. Since extrapolating 8 stages leads to very large entries in $P$, the predictor for the 8-stage methods was chosen to be the last step value predictor. If $\widehat{D}^{(mr)} > 4$, then this is indicated by $*$.

The $\widehat{D}^{(mr)}$-values for BDF are independent of $mr$, because for the linear test problem the corrector equation is solved within 1 iteration. For $k = 1,2,3$ and 4 these values are 0, 0, 0.0833 and 0.6665, respectively.

From these tables we see that for $s \leq 4$ the stability of PILSMRK($L$,$I$) is better than that of PILSMRK($L$,$U$). For $s = 8$ the $\widehat{D}$-values are comparable. Relatively to its order, the stability of PILSMRK is much better than that of BDF. As expected, we see that increasing $s$ and decreasing $k$ improves the

Table 3: Values of $\widehat{D}^{(mr)}$ for PILSMRK($L$,$I$) with $k$ steps and $s$ stages.

| $s$ | $k$ | $mr=1$ | $mr=2$ | $mr=4$ | $mr=6$ | $mr=8$ | $mr=10$ | $mr=20$ | $mr=\infty$ |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0.0094 | 0.0823 | 0.0838 | 0.0838 | 0.0838 | 0.0838 | 0.0838 |
| | 4 | 0 | 0.3435 | 0.4601 | 0.4610 | 0.4610 | 0.4610 | 0.4610 | 0.4610 |
| 4 | 1 | * | * | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | * | * | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | * | * | 0 | 0 | 0.0006 | 0.0021 | 0.0025 | 0.0025 |
| | 4 | * | * | 0 | 0 | 0.0120 | 0.0180 | 0.0192 | 0.0192 |
| 8 | 1 | 0 | 0 | 0.0677 | 0.0480 | 0.0239 | 0.0103 | 0 | 0 |
| | 2 | 0 | 0 | 0.0624 | 0.0405 | 0.0188 | 0.0076 | 0 | 0 |
| | 3 | 0 | 0 | 0.0590 | 0.0363 | 0.0162 | 0.0064 | 0 | 0 |
| | 4 | 0 | 0 | 0.0565 | 0.0335 | 0.0145 | 0.0057 | 0.0004 | 0.0003 |

Table 4: Values of $\widehat{D}^{(mr)}$ for PILSMRK($L$,$U$) with $k$ steps and $s$ stages.

| $s$ | $k$ | $mr=1$ | $mr=2$ | $mr=4$ | $mr=6$ | $mr=8$ | $mr=10$ | $mr=20$ | $mr=\infty$ |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0.0216 | 0.0827 | 0.0838 | 0.0838 | 0.0838 | 0.0838 | 0.0838 |
| | 4 | 0 | 0.3762 | 0.4605 | 0.4610 | 0.4610 | 0.4610 | 0.4610 | 0.4610 |
| 4 | 1 | * | * | 0.2214 | 0 | 0 | 0 | 0 | 0 |
| | 2 | * | * | 0.2239 | 0 | 0.0001 | 0 | 0 | 0 |
| | 3 | * | * | 0.2784 | 0 | 0.0031 | 0.0030 | 0.0025 | 0.0025 |
| | 4 | * | * | 0.3474 | 0.0001 | 0.0169 | 0.0194 | 0.0192 | 0.0192 |
| 8 | 1 | 0 | 0.1060 | 0.0636 | 0.0254 | 0.0212 | 0.0101 | 0 | 0 |
| | 2 | 0 | 0.1056 | 0.0557 | 0.0227 | 0.0179 | 0.0080 | 0 | 0 |
| | 3 | 0 | 0.1051 | 0.0510 | 0.0210 | 0.0161 | 0.0075 | 0 | 0 |
| | 4 | 0 | 0.1046 | 0.0477 | 0.0199 | 0.0152 | 0.0075 | 0.0003 | 0.0003 |

stability of MRK. If we solve the corrector equation only approximately, then sometimes the stability of the resulting method is even better than that of MRK. For $s = 4$ and $mr \leq 2$, the method is not stable, due to the extrapolation predictor, which is very unstable as stand-alone method. Notice that the $\widehat{D}^{(\infty)}$-values are the values for the underlying MRK corrector.

To get an idea of the shape of $\mathcal{S}^{(mr)}$, Figure 3 shows $\mathcal{S}^{(mr)}$ for PILSMRK-($L$,$U$) with 3 steps and 4 stages, where $mr \in \{3, 5, \infty\}$.

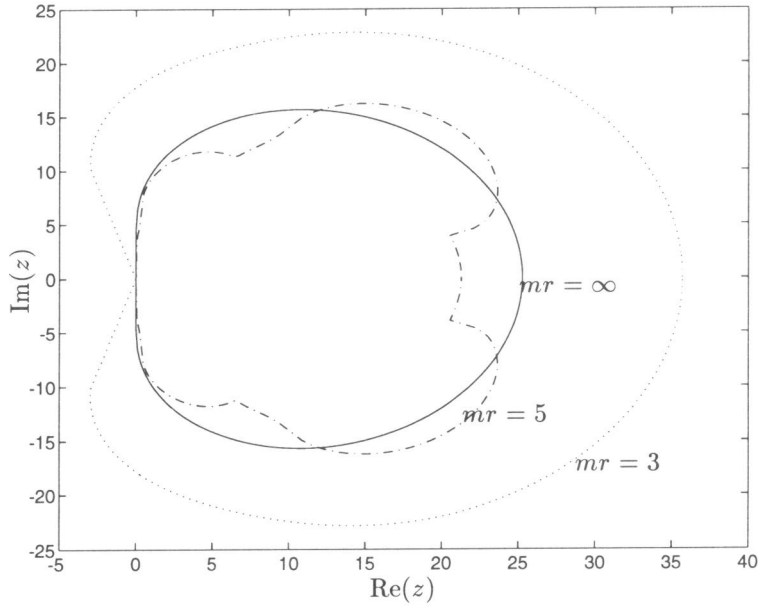


Figure 3: $\mathcal{S}^{(mr)}$ for PILSMRK($L$,$U$) with $k = 3, s = 4$.

# 5 Numerical experiments

In this paragraph we compare several Newton-PILSMRK methods with BDF. We also investigate how many inner iterations PILSMRK needs to find the Newton iterate. Although in practice one would use variable stepsizes and variable order, for these purposes it is sufficient to conduct experiments with fixed stepsize and fixed values of $s$ and $k$.

Two problems from the 'Test Set for IVP Solvers' [16] are integrated. Our first test example is a problem of Schäfer (called the HIRES problem in [9, p.157]) and consists of 8 mildly-stiff non-linear equations on the interval $[5, 305]$. (We adapted the initial condition here such that the integration starts outside the transient phase.) We used stepsize $h = 15$. The second test problem originates form circuit analysis and describes a ring modulator. We integrate this highly stiff system of 15 equations on the interval $[0, 10^{-3}]$ with stepsize $h = 2.5 \cdot 10^{-7}$. Horneber [11] provided this problem.

For $s > 1$ we implemented the extrapolation predictor as defined before, i.e. based on the previous stage vector. For BDF we used the last steppoint value as predictor. We tried extrapolation of more steppoints, but this did not give satisfactory results for both test problems. The starting values $y_1, y_2, \ldots, y_{k-1}$

Table 5: Results of PILSMRK($L$,$I$) for HIRES.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 3.3 | 3.7 | 4.2 | 4.7 | 4.9 |
| | | 2 | 3.2 | 3.8 | 4.3 | 5.0 | 4.9 |
| | | 10 | 3.2 | 3.8 | 4.3 | 5.0 | 4.9 |
| | 3 | 1 | 3.3 | 3.7 | 4.2 | 4.6 | 5.2 |
| | | 2 | 3.2 | 3.8 | 4.3 | 4.8 | 5.2 |
| | | 10 | 3.2 | 3.8 | 4.3 | 4.8 | 5.2 |
| 4 | 2 | 1 | * | 4.6 | 4.8 | 5.1 | 7.3 |
| | | 2 | * | 4.3 | 4.9 | 5.3 | 7.9 |
| | | 10 | 3.7 | 4.4 | 4.9 | 5.4 | 7.9 |
| | 3 | 1 | * | 4.6 | 4.8 | 5.1 | 7.2 |
| | | 2 | * | 4.3 | 4.9 | 5.3 | 7.8 |
| | | 10 | 3.7 | 4.4 | 4.9 | 5.4 | 7.8 |

Table 6: Results of PILSMRK($L$,$I$) for Ring modulator.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | * | 2.8 | 3.9 | 3.8 | 3.8 |
| | | 2 | * | 3.6 | 3.8 | 3.8 | 3.8 |
| | | 10 | * | 3.6 | 3.8 | 3.8 | 3.8 |
| | 3 | 1 | * | 3.0 | 4.1 | 4.2 | 4.3 |
| | | 2 | * | 3.8 | 4.2 | 4.3 | 4.3 |
| | | 10 | * | 3.8 | 4.2 | 4.3 | 4.3 |
| 4 | 2 | 1 | * | * | 6.1 | 6.5 | 8.2 |
| | | 2 | * | * | 5.8 | 6.5 | 8.2 |
| | | 10 | * | * | 5.8 | 6.4 | 8.2 |
| | 3 | 1 | * | * | 6.1 | 6.5 | 8.1 |
| | | 2 | * | * | 5.8 | 6.5 | 8.1 |
| | | 10 | * | * | 5.8 | 6.4 | 8.1 |

were obtained using the 8-stage Radau IIA method, in order to be sure that the integration is not influenced by some starting procedure. In the implementation of BDF we solved the non-linear equation of dimension $d$ with modified Newton, using $m$ iterations per time step.

In the tables we list the minimal number of correct digits $cd$ of the components of the numerical solution in the endpoint, i.e. at the endpoint, the absolute errors are written as $10^{-cd}$. Negative $cd$-values are indicated with $*$. The numbers of stages, steps, inner and outer iterations are given by $s$, $k$, $r$ and $m$, respectively.

The tables clearly show that the PILSMRK iterates for $r = 1$ are (almost) of the same quality as the Newton iterates, provided that we perform more

Table 7: Results of PILSMRK($L$,$U$) for HIRES.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 3.3 | 3.8 | 4.2 | 4.8 | 4.9 |
| | | 2 | 3.2 | 3.8 | 4.3 | 5.0 | 4.9 |
| | | 10 | 3.2 | 3.8 | 4.3 | 5.0 | 4.9 |
| | 3 | 1 | 3.3 | 3.8 | 4.2 | 4.7 | 5.2 |
| | | 2 | 3.2 | 3.8 | 4.3 | 4.8 | 5.2 |
| | | 10 | 3.2 | 3.8 | 4.3 | 4.8 | 5.2 |
| 4 | 2 | 1 | * | * | 4.9 | 5.1 | 7.2 |
| | | 2 | 2.6 | 4.4 | 4.9 | 5.4 | 7.9 |
| | | 10 | 3.7 | 4.4 | 4.9 | 5.4 | 7.9 |
| | 3 | 1 | * | * | 4.9 | 5.2 | 7.2 |
| | | 2 | 3.6 | 4.4 | 4.9 | 5.4 | 7.8 |
| | | 10 | 3.7 | 4.4 | 4.9 | 5.4 | 7.8 |

Table 8: Results of PILSMRK($L$,$U$) for Ring modulator.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | * | 2.8 | 3.9 | 3.8 | 3.8 |
| | | 2 | * | 3.6 | 3.8 | 3.8 | 3.8 |
| | | 10 | * | 3.6 | 3.8 | 3.8 | 3.8 |
| | 3 | 1 | * | 3.1 | 4.1 | 4.3 | 4.3 |
| | | 2 | * | 3.8 | 4.2 | 4.3 | 4.3 |
| | | 10 | * | 3.8 | 4.2 | 4.3 | 4.3 |
| 4 | 2 | 1 | * | * | 5.8 | 6.3 | 8.2 |
| | | 2 | * | * | 5.8 | 6.4 | 8.2 |
| | | 10 | * | * | 5.8 | 6.4 | 8.2 |
| | 3 | 1 | * | * | 5.8 | 6.3 | 8.1 |
| | | 2 | * | * | 5.8 | 6.4 | 8.1 |
| | | 10 | * | * | 5.8 | 6.4 | 8.1 |

than 2 Newton iterations. We also see that Newton-PILSMRK reaches higher accuracies than BDF for the same number of Newton iterations. However, if we want to solve the corrector equation entirely, one would have to perform more Newton iterations for Newton-PILSMRK than for BDF, since the latter is of lower order. Solving the ring modulator, BDF suffers from stability problems for $k = 6$, whereas the methods with $k \leq 4$ give *cd*-values, that might be too low in practice. For the 4-stage Newton-PILSMRK, the $k = 3$ results are not better than the $k = 2$ results. Performing not more then 10 Newton iterations, which is not sufficient to solve the corrector equation, is responsible for this. Experiments confirmed that using more than 10 iterations for the 3-step 4-stage MRK yields higher accuracies than for the 2-step 4-stage method.

Table 9: Results of BDF for HIRES.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2.9 | 3.5 | 3.1 | 3.0 | 3.0 |
| | 3 | 1 | 2.8 | 3.7 | 3.6 | 3.4 | 3.3 |
| | 4 | 1 | 2.8 | 3.4 | 4.4 | 3.8 | 3.6 |
| | 5 | 1 | 2.7 | 3.3 | 4.2 | 4.1 | 3.8 |
| | 6 | 1 | 2.8 | 3.4 | 4.1 | 3.9 | 3.7 |

Table 10: Results of BDF for Ring modulator.

| $s$ | $k$ | $r$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=10$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 1.1 | 1.1 | 1.1 | 1.1 | 1.1 |
| | 3 | 1 | 1.6 | 1.5 | 1.6 | 1.6 | 1.6 |
| | 4 | 1 | 1.8 | 1.9 | 1.9 | 1.9 | 1.9 |
| | 5 | 1 | 2.4 | 2.9 | 2.9 | 2.9 | 2.9 |
| | 6 | 1 | 2.4 | * | 2.9 | * | 2.9 |

A comparison of Tables 5 and 6 with Tables 7 and 8 shows that the performance of PILSMRK($L$,$U$) is comparable to that of PILSMRK($L$,$I$). Although PILSMRK($L$,$U$) converges faster than PILSMRK($L$,$I$), the latter has better stability properties for $s \leq 4$. Apparently, these effects neutralize each other for these test problems. However, Tables 1-4 indicate that PILSMRK($L$,$U$) can become better than PILSMRK($L$,$I$) for $s > 4$.

In order to show how the Newton-PILSMRK method performs on an $s$-processor computer, we implemented the 3-step 4-stage Newton PILSMRK($L$,$I$) on the Cray C-98/4256 at SARA and integrated the ring modulator, using again 4000 constant integration steps. The Cray C98/4256 is a shared memory computer with four processors. Table 11 lists the speed-up factors of the runs on four processors with respect to the runs in one-processor mode. Since we did not have the machine in dedicated mode during our experiments (on the average we used 2.5 processors concurrently), we used a tool called ATExpert [6] to predict the actual speed-up factors on four processors. In practice these values turn out to be very reliable. Denoting the fraction of the code that can be done in parallel by $f_P$, the optimal speed-up on $N$ processors according to Amdahl's law is given by the formula $1/(1 - f_P + f_P/N)$. ATExpert produces these optimal speed-up values, based on estimates of the parallel fraction $f_P$. These values are also listed in Table 11.

We compiled the codes using the flags `-dp`, `-ZP` and `-Wu"-p"`. The environment variables `NCPUS` and `MP_DEDICATED` were valued 4 and 1, respectively. We refer to the Cray C90 documentation [5] for the specification of these settings.

Table 11: Speed-up factor of 3-step 4-stage Newton-PILSMRK($L$,$I$) for ring modulator.

| | $m = 3$ | $m = 4$ | $m = 10$ |
|---|---|---|---|
| Actual speed-up | 3.3 | 3.3 | 3.2 |
| Optimal speed-up | 3.9 | 3.9 | 3.9 |

From Table 11 we conclude that the Newton-PILSMRK methods have a satisfactory parallel performance.

# 6 Summary and conclusions

In this paper we proposed the Newton-PILSMRK method, which is a combination of a Newton process applied to a Multistep Runge–Kutta method with a Parallel Iterative Linear System solver. The non-linear equations that arise in an MRK are usually solved by a modified Newton process, in which we have to solve linear systems of dimension $sd$, where $s$ is the number of Runge–Kutta stages of the MRK and $d$ the dimension of the problem. PILSMRK computes the solutions of these linear systems by means of an inner iteration process, in which we solve $s$ decoupled systems of dimension $d$. To achieve this decoupling, we have to approximate a matrix $A$ with complex eigenvalues by a matrix $B$ with positive distinct eigenvalues. It turns out that

- the most efficient parallel implementation of an MRK with a Newton process is 4 times more expensive than Newton-PILSMRK on $s$ processors in terms of $\mathcal{O}(d^3)$ costs.
- if we apply more than 2 Newton iterations, then in practice PILSMRK with only 1 inner iteration often suffices to find the Newton iterate,
- in terms of Jacobian evaluations and $LU$-decompositions, the $k$-step $s$-stage Newton-PILSMRK on $s$ processors is equally expensive as the $k$-step BDF on 1 processor, whereas the order is higher and the stability properties are better than that of BDF,
- for the same number of function evaluations, Newton-PILSMRK delivers higher accuracies than BDF, although Newton-PILSMRK did not solve the corrector equation entirely,
- increasing the number of previous steppoints $k$, leads to a better convergence behaviour of PILSMRK, but worse stability properties of the

MRK,

- in a linear stability analysis, performing more than 3 iterations (inner or outer) suffices to attain at least the stability of the MRK corrector, if $s \leq 4$.
- of the two options proposed here for choosing the matrix $B$, Crout and Schur-Crout, the latter has a better convergence behaviour, but its stability properties are worse for $s \leq 4$.

## Acknowledgements

The authors are grateful to Prof. dr. P.J. van der Houwen for his careful reading of the manuscript and for suggesting several improvements.

# Appendix

In this appendix we list the parameters $c$, $G$ and $A$ in (3) for the $k$-step $s$-stage MRK method of Radau type for $k \in \{2,3\}$ and $s \in \{2,4\}$. Moreover, we provide the PILSMRK parameters $\delta$ and $Q$, where $\delta$ =diag($D$) and $D$, $Q$ are the matrices in (9), for both the Crout approach (i.e. PILSMRK($L$,$I$)) and the Schur-Crout approach (i.e. PILSMRK($L$,$U$)).

$s = 2,\ k = 2:$

$$c^T = \begin{bmatrix} 0.39038820320221 & 1.00000000000000 \end{bmatrix}$$

$$G = \begin{bmatrix} -0.04671554852736 & 1.04671554852736 \\ -0.02010509586877 & 1.02010509586877 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.40044075113659 & -0.05676809646175 \\ 0.77072385847003 & 0.20917104566120 \end{bmatrix}$$

Crout:

$$\delta^T = \begin{bmatrix} 0.40044075113659 & 0.31843196932797 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1.00000000000000 & 0 \\ 9.39806495685529 & 1.00000000000000 \end{bmatrix}$$

Schur-Crout:

$$\delta^T = \begin{bmatrix} 0.36028586267747 & 0.35392212182843 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.06418485435680 & 0.05604152383747 \\ 0.99793802636797 & 0.99842843890084 \end{bmatrix}$$

$s = 2,\ k = 3:$

$$c^T = \begin{bmatrix} 0.42408624230810 & 1.00000000000000 \end{bmatrix}$$

$$G = \begin{bmatrix} 0.01290709720739 & -0.10843463813621 & 1.09552754092881 \\ 0.00354588047065 & -0.04623386039657 & 1.04268797992593 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.38745055226697 & -0.04598475368028 \\ 0.77239469511979 & 0.18846320542493 \end{bmatrix}$$

Crout:

$$\delta^T = \begin{bmatrix} 0.38745055226697 & 0.28013523838816 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1.00000000000000 & 0 \\ 7.19743219492460 & 1.00000000000000 \end{bmatrix}$$

Schur-Crout:

$$\delta^T = \begin{bmatrix} 0.33129449207677 & 0.32761955124138 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.08083975113162 & 0.07616492879483 \\ 0.99672711141866 & 0.99709523297510 \end{bmatrix}$$

$s = 4,\ k = 2$ :

$$c^T = \begin{bmatrix} 0.09878664634426 & 0.43388702543882 & 0.80169299888049 & 1.00000000000000 \end{bmatrix}$$

$$G = \begin{bmatrix} -0.00087353889029 & 1.00087353889029 \\ 0.00062121019919 & 0.99937878980081 \\ -0.00032939714868 & 1.00032939714868 \\ -0.00003663563426 & 1.00003663563426 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.11996670457577 & -0.03384322082318 & 0.01835753398261 & -0.00656791028123 \\ 0.26010642038045 & 0.20159324902943 & -0.03956525951247 & 0.01237382574059 \\ 0.23561500946812 & 0.41088455735437 & 0.17597260265111 & -0.02110856774179 \\ 0.24141835002666 & 0.38984924120599 & 0.31101721961059 & 0.05767855352250 \end{bmatrix}$$

Crout:

$$\delta^T = \begin{bmatrix} 0.10617138884400 & 0.27770096849016 & 0.27497060030028 & 0.11996670457577 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0.01481904140434 \\ 0 & 0 & 0.00220678551539 & -0.02486729636785 \\ 0 & 0.38896861370956 & -0.38581432071631 & 0.05312025222596 \\ 1.00000000000000 & 0.92125100681023 & -0.92257381278026 & 0.99816844890362 \end{bmatrix}$$

Schur-Crout:

$$\delta^T = \begin{bmatrix} 0.17879165196884 & 0.15567835604316 & 0.18725864804630 & 0.18660124038403 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.05047735457027 & -0.05698986733483 & 0.04780729735701 & 0.04801402184956 \\ 0.17096598389037 & 0.17933373843615 & -0.16592112501907 & -0.16634392504346 \\ -0.15139062605533 & -0.08731023751861 & 0.17251778528806 & 0.17091936180350 \\ -0.97226722014607 & -0.97824766174222 & 0.96975370911955 & 0.96995408348419 \end{bmatrix}$$

$s = 4,\ k = 3$ :

$$c^T = \begin{bmatrix} 0.10504182884419 & 0.44825417107884 & 0.80977028814179 & 1.00000000000000 \end{bmatrix}$$

$$G = \begin{bmatrix} 0.00007487445528 & -0.00195646912651 & 1.00188159467123 \\ -0.00007345206497 & 0.00148038414152 & 0.99859306792346 \\ 0.00003966973124 & -0.00083011136249 & 1.00079044163125 \\ 0.00000077039880 & -0.00008665832447 & 1.00008588792568 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.12388725564952 & -0.03052720746880 & 0.01502960651127 & -0.00515454606376 \\ 0.27600575210564 & 0.19832624728391 & -0.03534802573852 & 0.01060367743938 \\ 0.24659262259186 & 0.41336961213203 & 0.16850574024079 & -0.01944845872291 \\ 0.25397302181219 & 0.39037260118042 & 0.30064393200968 & 0.05492532747083 \end{bmatrix}$$

Crout:

$$\delta^T = \begin{bmatrix} 0.09980104557325 & 0.26112476902731 & 0.26633715617793 & 0.12388725564952 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0.01885332656568 \\ 0 & 0 & 0.00429974732457 & -0.03652952061848 \\ 0 & 0.38485479574542 & 0.39111661588660 & 0.09232717942550 \\ 1.00000000000000 & 0.92297713199827 & 0.92033108442036 & 0.99487981090186 \end{bmatrix}$$

Schur-Crout:

$$\delta^T = \begin{bmatrix} 0.17281106755693 & 0.15348751145786 & 0.18030166423062 & 0.17980311756297 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.03747602767829 & -0.04253832729434 & 0.03538720965186 & 0.03552524964325 \\ 0.15438230915055 & 0.16281308949598 & -0.14969201305536 & -0.15002487334226 \\ -0.16907928673314 & -0.11582715575719 & 0.18786790085145 & 0.18664755906307 \\ -0.97271467798559 & -0.97891085323893 & 0.97007509938672 & 0.97025418458887 \end{bmatrix}$$

## References

[1] Zhou Bing. $A$-stable and $L$-stable block implicit one-block methods. *Journal of Computational Mathematics*, 3(4):328–341, 1985.

[2] Peter N. Brown, Alan C. Hindmarsh, and George D. Byrne. *VODE: A variable coefficient ODE solver*, August 1992. Available via WWW at URL http://www.netlib.org/ode/vode.f.

[3] J. C. Butcher. On the implementation of implicit Runge–Kutta methods. *BIT*, 16:237–240, 1976.

[4] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, New York, 1991.

[5] Cray Research, Inc. *CF77 Commands and Directives*, SR-3771 6.0 edition, 1994.

[6] Cray Research, Inc. *UNICOS Performance Utilities Reference Manual*, SR-2040 8.0 edition, 1994.

[7] A. Guillou and J. L. Soulé. La résolution numérique des problèmes différentiels aux conditions initiales par des méthodes de collocation. *R.I.R.O.*, R-3:17–44, 1969.

[8] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, second revised edition, 1993.

[9] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems.* Springer-Verlag, 1991.

[10] E. Hairer and G. Wanner. *RADAU5*, July 1996. Available via WWW at URL ftp://ftp.unige.ch/pub/doc/math/stiff/radau5.f.

[11] E. H. Horneber. *Analyse nichtlinearer RLCÜ-Netzwerke mit Hilfe der gemischten Potentialfunktion mit einer systematischen Darstellung der Analyse nichtlinearer dynamischer Netzwerke.* PhD thesis, Universität Kaiserslautern, 1976.

[12] P. J. van der Houwen and E. Messina. Parallel linear system solvers for Runge–Kutta–Nyström methods. Technical Report NM-R9613, CWI, Amsterdam, 1996. Submitted for publication.

[13] P. J. van der Houwen and J. J. B. de Swart. Parallel linear system solvers for Runge–Kutta methods. *Advances in Computational Mathematics*, 7:157–181, 1997.

[14] P. J. van der Houwen and J. J. B. de Swart. Triangularly implicit iteration methods for ODE-IVP solvers. *SIAM J. Sci. Comput.*, 18(1):41–55, 1997.

[15] I. Lie and S. P. Nørsett. The stability function for multistep collocation methods. *Numer. Math.*, 57:779–787, 1989.

[16] W. M. Lioen, J. J. B. de Swart, and W. A. van der Veen. Test set for IVP solvers. Report NM-R9615, CWI, Amsterdam, 1996. WWW version available at URL http://www.cwi.nl/cwi/projects/IVPtestset.shtml.

[17] O. Nevanlinna. Matrix valued versions of a result of Von Neumann with an application to time discretization. *J. Comput. Appl. Math.*, 12 & 13:475–489, 1985.

[18] L. R. Petzold. *DASSL: A Differential/Algebraic System Solver*, June 1991. Available via WWW at URL http://www.netlib.org/ode/ddassl.f.

[19] S. Schneider. *Intégration de systèmes d'équations différentielles raides et différentielles-algébriques par des méthodes de collocations et méthodes générales linéaires.* PhD thesis, Université de Genéve, 1994.

# Chapter 7

# The Solution of Implicit Differential Equations on Parallel Computers

P.J. van der Houwen & W.A. van der Veen

**Abstract**

We construct and analyse parallel iterative solvers for the solution of the linear systems arising in the application of Newton's method to s-stage implicit Runge-Kutta (RK) type discretizations of implicit differential equations (IDEs). These linear solvers are partly iterative and partly direct. Each linear system iteration again requires the solution of linear subsystems, but now only of IDE dimension, which is s times less than the dimension of the linear systems in Newton's method. Thus, the effective costs on a parallel computer system are only one LU-decomposition of IDE dimension for each Jacobian update, yielding a considerable reduction of the effective LU costs. The method parameters can be chosen such that only a few iterations by the linear solver are needed. The algorithmic properties are illustrated by solving the transistor problem (index 1) and the car axis problem (index 3) taken from the CWI test set.

*CR Subject Classification (1991):* G.1.7
*Keywords and Phrases:* numerical analysis, implicit ODEs, DAEs, Runge-Kutta methods, parallelism.

*Note:* The research reported in this paper was partiallly supported by the Technology Foundation (STW) in the Netherlands.

## 1. Introduction

We consider initial value problems (IVPs) for systems of implicit differential equations (IDEs)

$$(1.1) \qquad \phi(\dot{\mathbf{y}}(t),\mathbf{y}(t)) = \mathbf{0}, \qquad \mathbf{y}, \phi \in \mathbb{R}^d.$$

It will be assumed that the initial conditions are consistent and that the IVP has a unique solution. Furthermore, we define the Jacobian matrices $K := \phi_{\mathbf{u}}(\mathbf{u},\mathbf{v})$ and $J := -\phi_{\mathbf{v}}(\mathbf{u},\mathbf{v})$. In the case of *explicit* differential equations $\phi = \mathbf{y}' - \mathbf{f}(t,\mathbf{y})$, J denotes the Jacobian of the righthand side function.

In this paper, we construct and analyse parallel iterative solvers for the solution of the sd-dimensional linear systems arising in the application of Newton's method to s-stage implicit Runge-Kutta (RK) type discretizations of (1.1). These linear solvers may be considered as *inner* iteration processes (the Newton process itself is the *outer* iteration process). The inner iteration process is partly an iterative method and partly a direct method, that is, each inner iteration again requires the solution of linear subsystems, but now only of dimension d. We assume that direct solution methods are used for solving these subsystems. Thus, the effective costs on a parallel computer system are only one LU-decomposition of IVP dimension for each Jacobian update, yielding a considerable reduction of the *effective* LU costs. As to the backward-forward substitutions (briefly FBSs) needed in each inner

iteration, we distinguish the Jacobi and the Gauss-Seidel approach. In the Jacobi approach, the s FBSs per inner iteration can be executed in parallel, so that both the LU costs and the FBS costs are independent of the number of stages used in the RK discretization. For ODEIVPs, this approach was used in [9] and [11]. We shall show that with appropriate changes, it can also be used in the IDE case. In the Gauss-Seidel approach, part of the FBSs per inner iteration have to be done sequentially which increases the effective FBS costs. We shall compare the convergence factors of the Jacobi and the Gauss-Seidel approach, taking into account the effect of the predictor.
The algorithmic properties of the inner iteration method are illustrated by solving an IDE of index 1 and of index 3 taken from the literature.

**2. Runge-Kutta discretization**

We define the Runge-Kutta type formula (cf. [2] or [4])

$$(2.1) \qquad \mathbf{y}_{n+1} = (\mathbf{e}_s^T \otimes I)\mathbf{Y}_{n+1}, \quad \mathbf{R}_n(\mathbf{Y}_{n+1}) = \mathbf{0}, \quad \mathbf{R}_n(\mathbf{Y}) := \Phi\big((h^{-1}A^{-1}\otimes I)(\mathbf{Y} - \mathbf{W}_n), \mathbf{Y}\big).$$

Here, A denotes the s-by-s RK matrix which is assumed to be nonsingular, $\mathbf{W}_n$ is an sd-dimensional vector containing information from preceding steps, I is the d-by-d identity matrix, h is the stepsize $t_{n+1} - t_n$, and $\otimes$ denotes the Kronecker product. The s vector components $\mathbf{Y}_{n+1,i}$ of the sd-dimensional solution vector $\mathbf{Y}_{n+1}$ represent numerical approximations to the exact solution vectors $\mathbf{y}(t_n + c_i h)$, $\mathbf{c} = (c_i)$ being the abscissas vector with $c_s = 1$. Furthermore, $\mathbf{e}_s$ is the sth unit vector, $\mathbf{y}_n$ is the numerical approximation to $\mathbf{y}(t_n)$, and $\Phi(\mathbf{U},\mathbf{V})$ contains the values $\big(\phi(\mathbf{U}_i,\mathbf{V}_i)\big)$ for any pair of vectors $\mathbf{U} = \big(\mathbf{U}_i\big)$ and $\mathbf{V} = \big(\mathbf{V}_i\big)$. In the following, I will denote the identity matrix, the dimension of which will be clear from the context.
The method (2.1) is completely specified by the matrix pair $\{A, \mathbf{W}_n\}$. If $\mathbf{W}_n = (E\otimes I)\mathbf{Y}_n$ with $E := \mathbf{e}\mathbf{e}_s^T$, $\mathbf{e}_s^T$ and $\mathbf{e}$ representing the sth unit vector and the s-dimensional vector with unit entries, then (2.1) represents the one-step RK method $\{A, \mathbf{b}, \mathbf{c}\} = \{A, A^T\mathbf{e}_s, \mathbf{c}\}$. Alternatives are the block methods where E is a matrix with eigenvalues on the unit disk, those of magnitude 1 being simple, and the k-step Radau methods where $\mathbf{W}_n$ is defined by a linear combination of the step point values $\mathbf{y}_n, \mathbf{y}_{n-1}, \ldots, \mathbf{y}_{n-k+1}$ (see e.g. [4]). In the following, most of our analysis applies to general back information vectors $\mathbf{W}_n$, but numerical illustrations will be confined to one-step Radau IIA methods.
The usual approach for solving the implicit equation $\mathbf{R}_n(\mathbf{Y}) = \mathbf{0}$ in (2.1) is the application of the modified Newton method

$$(2.2) \qquad N_A\big(\mathbf{Y}^{(j)} - \mathbf{Y}^{(j-1)}\big) = -(hA\otimes I)\mathbf{R}_n(\mathbf{Y}^{(j-1)}), \quad N_A := I\otimes K_n - A\otimes hJ_n, \quad j = 1, \ldots, m,$$

where $K_n$ and $J_n$ are the Jacobian matrices K and J evaluated at the step point $t_n$. Each Newton iteration requires the solution of a linear system of dimension sd. However, already for moderate values of d, this is quite expensive, because the LU-decomposition of the sd-by-sd matrix $N_A$ requires as many as $\frac{2}{3}s^3d^3$ arithmetic operations. This number of operations can be reduced by transforming (2.2) to 'block-diagonal' form (cf. Butcher [3]). Let $\mathbf{Y}^{(j)} = (Q\otimes I)\widetilde{\mathbf{Y}}^{(j)}$, then (2.2) transforms to

$$\text{(2.2')} \qquad \begin{aligned} &\widetilde{N}_A\big(\widetilde{\mathbf{Y}}^{(j)} - \widetilde{\mathbf{Y}}^{(j-1)}\big) = -\,(hQ^{-1}A\otimes I)\mathbf{R}_n((Q\otimes I)\widetilde{\mathbf{Y}}^{(j-1)}), \quad j = 1, \ldots, m, \\ &\widetilde{N}_A := (Q^{-1}\otimes I)N_A(Q\otimes I) = I\otimes K_n - \widetilde{A}\otimes hJ_n, \quad \widetilde{A} = Q^{-1}AQ. \end{aligned}$$

Assuming that A is nondefect, we can choose Q such that $\widetilde{A}$ is a $\sigma$-by-$\sigma$ block-diagonal with either one-by-one or two-by-two, *real* diagonal blocks, where each diagonal block corresponds to an eigenvalue (pair) $\xi_k \pm i\eta_k$ of A. In fact, if $\eta_k = 0$, then $\widetilde{A}_{kk} = \xi_k$, and

$$\text{(2.3)} \qquad \widetilde{A}_{kk} = \begin{pmatrix} a_k & b_k \\ c_k & 2\xi_k - a_k \end{pmatrix}, \quad b_kc_k = -\,(a_k^2 - 2\xi_k a_k + \alpha_k^2), \quad \alpha_k := \sqrt{\xi_k^2 + \eta_k^2},$$

otherwise (here $a_k$, $b_k$ and $c_k$ are real parameters which depend on the matrix Q). In the following it will always be assumed that $\xi_k > 0$ and that the ordering of the diagonal blocks $\widetilde{A}_{kk}$ is such that the ratio $|\eta_k / \xi_k|$ increases with k.

If the RK matrix A has only real eigenvalues, as is the case in the methods designed by Orel [12] and Bendtsen [1], then all diagonal blocks of $\widetilde{N}_A$ are of order d. When solving the linear system in (2.2') we need the LU-decompositions of these diagonal blocks, requiring $\frac{2}{3}d^3$ operations. Hence, the *total* LU-costs are $\frac{2}{3}sd^3$ operations. However, since the LU-decompositions can be computed concurrently, the *effective* computational LU-costs in the block-diagonalized Newton method (2.2') are only $\frac{2}{3}d^3$ operations, irrespective the value of s. Similarly, the FBSs can be performed in parallel.

A drawback of RK matrices with real eigenvalues is the relatively large value of s (and hence large numbers of processors) in order to achieve A-stability. More powerful methods with respect to order of accuracy and stability can be obtained by allowing A to have complex eigenvalues. For example, in the case of RK discretizations based on a Gaussian quadrature formulas, A has atmost one real eigenvalue (if s is odd). Hence, the diagonal blocks of $\widetilde{N}_A$ are of order either d or 2d, so that the LU-decompositions require either $\frac{2}{3}d^3$ or $\frac{16}{3}d^3$ operations. However, by writing the 2d-dimensional systems as d-dimensional systems with complex coefficients, the LU costs can be reduced to $\frac{8}{3}d^3$ operations (cf. Hairer and Wanner [4]). Then, the *total* LU-costs are $\frac{2}{3}(2s - 1)d^3$ operations for s odd and $\frac{2}{3}(2s)d^3$ operations for s even. Again, the LU-decompositions can be computed concurrently, so that the *effective* computational LU-costs are only $\frac{8}{3}d^3$ operations, irrespective the value of s. The RADAUP codes of Hairer and Wanner [5] use the block-diagonalized Newton method applied to Radau IIA discretizations of (1.1).

**Remark 2.1.** In practice, it may be recommendable to remove the $h^{-1}$ factor occurring in the residual function $\mathbf{R}_n(\mathbf{Y}^{(j-1)})$ by introducing 'derivative' iterates $\dot{\mathbf{Y}}^{(j)}$ by the relation $\mathbf{Y}^{(j)} = \mathbf{W}_n + h(A\otimes I)\dot{\mathbf{Y}}^{(j)}$. Then, the iteration scheme (2.2) becomes

$$\text{(2.4b)} \qquad N_A\big(\dot{\mathbf{Y}}^{(j)} - \dot{\mathbf{Y}}^{(j-1)}\big) = -\,\mathbf{R}_n(\mathbf{W}_n + h(A\otimes I)\dot{\mathbf{Y}}^{(j-1)}), \quad j = 1, \ldots, m,$$

The sequences $\{\mathbf{Y}^{(j)}\}$ generated by the schemes (2.2) and (2.4) are algebraically identical, but (2.4) can be used as $h \to 0$. Furthermore, the structure of the Newton equations in (2.2) and (2.4b) is similar and only differs by the righthand side. ♦

**3. Parallel linear system solvers**

The two main numerical approaches for solving the linear systems in (2.2) are the fully direct approach and a fully iterative approach. In this paper, we propose a combination of an iterative and a direct approach. To describe this method, we start with the *block-triangularized* Newton method, that is, in (2.2') the matrix Q is such that $\widetilde{A}$ now is $\sigma$-by-$\sigma$ *block-triangular* with *real* diagonal blocks. These blocks are again defined as in (2.3) and the ordering of the diagonal blocks $\widetilde{A}_{kk}$ is again such that $|\eta_k / \xi_k|$ increases with k. Next we split the block-diagonal part of the matrix $\widetilde{A}$ occurring in (2.2'). Let $\widetilde{C}$ be the strictly lower block-triangular part of $\widetilde{A}$, let $\widetilde{B}$ be a nondefective, $\sigma$-by-$\sigma$ block-diagonal matrix with positive eigenvalues whose one-by-one blocks equal $\xi_k$, and consider the iterative method

$$(3.1)\qquad \widetilde{N}_B\big(\widetilde{\mathbf{Y}}^{(j,\nu)} - \widetilde{\mathbf{Y}}^{(j,\nu-1)}\big) = (\widetilde{C}\otimes hJ_n)\big(\widetilde{\mathbf{Y}}^{(j)} - \widetilde{\mathbf{Y}}^{(j,\nu-1)}\big) + \widetilde{N}_A\big(\widetilde{\mathbf{Y}}^{(j-1)} - \widetilde{\mathbf{Y}}^{(j,\nu-1)}\big)$$
$$- (hQ^{-1}A\otimes I)\mathbf{R}_n((Q\otimes I)\widetilde{\mathbf{Y}}^{(j-1)}),\ \nu = 1, 2, \ldots, r,$$

where $\widetilde{N}_B$ is defined in the same way as $\widetilde{N}_A$ (see (2.2')). The iteration processes (2.2') and (3.1) may be considered as an *outer* and an *inner* iteration method, respectively with iteration indices j and $\nu$. The reason for using a block-triangular matrix $\widetilde{A}$ is that well-conditioned transformation matrices Q can be chosen such that there is no danger for amplification of the iteration errors $\widetilde{\mathbf{Y}}^{(j,\nu)} - \widetilde{\mathbf{Y}}^{(j)}$ by an ill-conditioned back transformation.

The method (3.1) consists of the *sequential* application of $\sigma$ iteration processes. In each iterative subprocess, we have to solve subsystems with matrix of coefficients $I\otimes K_n - \widetilde{B}_{kk}\otimes hJ_n$, where $\widetilde{B}_{kk}$ is a diagonal block of $\widetilde{B}$ and $k = 1, \ldots, \sigma$. We remark that only one iteration is required if $\widetilde{B}_{kk}$ is a one-by-one block, because then $\widetilde{B}_{kk} = \widetilde{A}_{kk} = \xi_k$. In fact, the number of inner iterations may differ for each subsystem and the quantity r in (3.1) should be interpreted as the *maximal* number of iterations needed for solving the subsystems. Furthermore, note that in the case of *block-triangular* matrices $\widetilde{A}$, the nonzero term $(\widetilde{C}\otimes hJ_n)\widetilde{\mathbf{Y}}^{(j)}$ implies that the subsystems should be iterated until convergence, otherwise $\widetilde{\mathbf{Y}}^{(j)}$ is not available. In the case of *block-diagonal* matrices $\widetilde{A}$, i.e. $\widetilde{C} = O$, there is no need to iterate until convergence, and moreover, the $\sigma$ iterative subprocesses can be carried out simultaneously. The $\widetilde{C} = O$ and $\widetilde{C} \neq O$ version of the iteration method (3.1) may respectively be considered as Jacobi and Gauss-Seidel type methods.

Since $\widetilde{B}$ is nondefective, the iterative subprocesses can be diagonalized, so that only linear systems of dimension d are to be solved. Assuming that these d-dimensional subsystems are solved by a direct method, it follows that only LU-decompositions of matrices of order d are required. Moreover, these LU-decompositions can be done in parallel, so that the effective LU-costs are $\frac{2}{3}d^3$ operations (irrespective the value of s), yielding a reduction by a factor 4 when compared with the $\frac{8}{3}d^3$ operations required by the solution of the complex, d-dimensional subsystems in the block-triangularized Newton method (2.2'). The main sequential part consists of the sequential execution of the FBSs to solve the s subsystems of dimension d. If $\bar{r}$ is the *averaged* number of iterations needed to solve these subsystems, then $s\bar{r}$ linear system solves are required, i.e. $2s\bar{r}d^2$ operations. If s is even, then effectively $s\bar{r}d^2$ operations are required, whereas block-diagonalized Newton (with block-diagonal $\widetilde{A}$) effectively requires only $8d^2$ operations. Hence, if $\bar{r}$ is large, then the advantage of the

reduced LU costs is easily lost. Thus, the linear solver (3.1) is only effective if we can choose $\widetilde{B}$ such that $\bar{r}$ is small.

The linear solver (3.1) will be called a *PILSRK method* (parallel iterative linear system method for RK discretizations) and the process {(2.2'), (3.1)} will be called a *Newton-PILSRK method*.

Defining the pair $\{B, C\} = \{\widetilde{Q}B\widetilde{Q}^{-1}, \widetilde{Q}C\widetilde{Q}^{-1}\}$, the back transformation of (3.1) reads

$$(3.2) \qquad N_B\left(\mathbf{Y}^{(j,\nu)} - \mathbf{Y}^{(j,\nu-1)}\right) = (C\otimes hJ_n)\left(\mathbf{Y}^{(j)} - \mathbf{Y}^{(j,\nu-1)}\right) + N_A\left(\mathbf{Y}^{(j-1)} - \mathbf{Y}^{(j,\nu-1)}\right) - (hA\otimes I)\mathbf{R}_n(\mathbf{Y}^{(j-1)}), \quad \nu = 1, 2, \ldots, r.$$

Estimates of the speed of convergence should be based on the iteration errors associated with (3.2), rather than on the iteration errors associated with (3.1).

**Remark 3.1.** The method (3.2) is a consistent iteration process for solving the linear Newton systems in (2.2) for any pair {B, C} for which (i) B, C and A - C are respectively similar to a diagonal matrix with positive diagonal entries, a strictly lower triangular matrix and a block-diagonal matrix, under the *same* similarity transformation. and (ii) C and A - C have the same partitioning. Hence, we can define a more general family of PILSRK{B,C} methods. The diagonalized version of this generalization possesses the same amount of parallelism as (3.1). In this connection, we remark that if C is similar to a strictly lower *block-triangular* matrix (e.g. if $C = \widetilde{Q}C\widetilde{Q}^{-1}$), then a number of the FBSs can be done in parallel reducing the costs of the FBS part. In the Jacobi case where C = O, *all* FBSs can be executed in parallel.

**Remark 3.2.** The Jacobi version of (3.2) can be considered as a conventional iteration method for linear systems based on the splitting $N_A = (I\otimes K_n - B\otimes hJ_n) + (B - A)\otimes hJ_n$. ♦

**Remark 3.3.** If only one inner iteration is performed and if we set $\mathbf{Y}^{(j,0)} = \mathbf{Y}^{(j-1)}$ and $\mathbf{Y}^{(j)} = \mathbf{Y}^{(j,1)}$, then the PILSRK method (3.2) reduces to

$$(3.3) \qquad \left(I\otimes K_n - (B + C)\otimes hJ_n\right)\left(\mathbf{Y}^{(j)} - \mathbf{Y}^{(j-1)}\right) = - (hA\otimes I)\mathbf{R}_n(\mathbf{Y}^{(j-1)}).$$

This scheme may be considered as an 'approximate' Newton scheme obtained by replacing in (2.2) the matrix A by B + C. Since B + C is similar to a lower triangular matrix with positive diagonal entries, we can diagonalize (3.3), so that effectively only one FBS is required per outer iteration. The method (3.3) is related to the PDIRK and PTIRK methods proposed in [7] and [8] for ODEs. These PDIRK and PTIRK methods are obtained by replacing $K_n$ with the identity matrix, by setting C = O, and by choosing for B either a diagonal matrix or a lower triangular matrix. ♦

**Remark 3.4.** Even on *sequential* computers the diagonalized forms of the PILSRK methods (3.2) may be more efficient than the block-diagonalized Newton method. For example, if s is even, then the total LU-costs and FBS costs associated with (3.2) respectively require $\frac{2}{3}sd^3$ and $2s\bar{r}d^2$ operations, whereas block-diagonalized Newton requires $\frac{4}{3}sd^3$ and $4sd^2$ operations. Hence, if $\bar{r} \leq 2$, then the PILSRK method does not require more FBS costs, while its LU costs are 2 times less expensive. ♦

**4. Convergence results**

The convergence can be studied by analysing the (exact) error recursion

$$(4.1) \qquad \mathbf{Y}^{(j,\nu)} - \mathbf{Y}^{(j)} = M\left(\mathbf{Y}^{(j,\nu-1)} - \mathbf{Y}^{(j)}\right), \quad M := \left(I\otimes K_n - B\otimes hJ_n\right)^{-1}\left((A - B - C)\otimes hJ_n\right).$$

We recall that in the Gauss-Seidel version $C \neq O$ it is assumed that we iterate until convergence, because (3.2) contains the term $(C\otimes hJ_n)\mathbf{Y}^{(j)}$. Only under this assumption, the recursion (4.1) is valid. In the convergence analysis, we shall suppose that

(i) $K_n$ is nonsingular,
(ii) $\{K_n, J_n\}$ has a complete (generalized) eigensystem

(we will refer to these assumptions as property P). In practice, this is of course an unrealistic situation. However, by observing that d-by-d matrix pairs $\{K, J\}$ having property P are dense in the space of all d-by-d matrix pairs, we can define a one-parameter family of matrix pairs $\{K(\varepsilon), J(\varepsilon)\}$ which satisfies property P for $\varepsilon > 0$ and which converges to $\{K_n, J_n\}$ as $\varepsilon \to 0$. Hence, for the matrix $M(\varepsilon)$ corresponding to $\{K(\varepsilon), J(\varepsilon)\}$we have $M(\varepsilon) \to M$ as $\varepsilon \to 0$. Thus, a convergence analysis based on property P is relevant for the case where this property is not satisfied.
A necessary and sufficient condition for convergence of the PILSRK methods is $\rho(M) < 1$. In order to obtain the eigenvalues of M, we shall list all its eigenvectors. First we look for eigenvectors of the form $\mathbf{a}\otimes\mathbf{w}$. If the eigenvalues are denoted by $\mu$, then $h\,(A - B - C + \mu B)\mathbf{a}\otimes J_n\mathbf{w} = \mu\,\mathbf{a}\otimes K_n\mathbf{w}$. This shows that $J_n\mathbf{w}$ and $K_n\mathbf{w}$ are related by the eigenvalue equation $J_n\mathbf{w} = \lambda K_n\mathbf{w}$, i.e. $\lambda$ is a (generalized) eigenvalue of the matrix pair $\{K_n, J_n\}$. On substitution of $J_n\mathbf{w} = \lambda K_n\mathbf{w}$ and by defining $z := \lambda h$, we obtain

$$z(A - B - C)\mathbf{a}\otimes K_n\mathbf{w} = \mu\left(I - zB\right)\mathbf{a}\otimes K_n\mathbf{w}.$$

Since $K_n\mathbf{w} \neq \mathbf{0}$, $\mu = \mu(z)$ is an eigenvalue of the amplification matrix

$$(4.2) \qquad Z(z) := z\left(I - zB\right)^{-1}\left(A - B - C\right).$$

We now impose the condition that $Z(h\lambda)$ is nondefect for all $\lambda$ in $\sigma(K_n, J_n)$. This condition and condition (ii) of property P implies that M has sd eigenvectors of the form $\mathbf{a}\otimes\mathbf{w}$. Hence, *all* eigenvectors of M are of this form and its eigenvalues are given by those of $Z(h\lambda)$ with $\lambda \in \sigma(K_n, J_n)$.
Let $\rho(z)$ be the spectral radius of $Z(z)$. Then, the *region of convergence* is defined by the region $\mathbb{I}^a := \{z: \rho(z) < 1, Z(z) \text{ is nondefect}\}$. In analogy with the terminilogy used in the linear stability theory, we shall call the PILSRK method *A-convergent* if $\mathbb{I}^a$ contains the left halfplane and *L-convergent* if it is A-convergent and if $\rho(z)$ vanishes at infinity. Matrix pairs $\{B, C\}$ will be said to lie in the set $\mathbb{B}(A)$ if

(i) B, C and A - C are similar (under the same similarity transformation) to a diagonal matrix with positive diagonal entries, a strictly lower triangular matrix and a block-diagonal matrix,
(ii) $\{B, C\}$ generates an A-convergent PILSRK method.

The considerations above are summarized in the following convergence result:

**Theorem 4.1.** Let $\{B, C\} \in \Xi(A)$, let $\{K_n, J_n\}$ satisfy property P and let its eigenvalues be in the nonpositive halfplane. Then, the PILSRK method (3.2) converges for all $h > 0$. ♦

In order to get some insight into the convergence behaviour, we observe that after $\nu$ iterations the eigenvector components of the iteration error are amplified by $Z^\nu(z)$, where $z = h\lambda$ corresponds with the (generalized) eigenvectors of $\{K_n, J_n\}$. Let us define the *averaged amplification factor* by

$$\rho^{(\nu)}(z) := \sqrt[\nu]{\| Z^\nu(z) \|} \quad . \tag{4.3}$$

Evidently, the spectral radius $\rho(z)$ of $Z(z)$ equals $\rho^{(\infty)}(z)$ and will therefore be called the *asymptotic* amplification factor.

### 4.1. The asymptotic amplification factor

We shall now consider the case $\{B, C\} = \{Q\tilde{B}Q^{-1}, Q\tilde{C}Q^{-1}\}$, i.e the process (3.1), in more detail. We first derive estimates for the asymptotic amplification factor $\rho(z) = \rho^{(\infty)}(z)$. Since the matrix $\tilde{Z}(z) = QZ(z)Q^{-1}$ is block-diagonal with either one-by-one or two-by-two diagonal blocks, we may confine our considerations to the diagonal blocks of $\tilde{Z}(z)$. Let us define the one-by-one blocks $\tilde{B}_{kk}$ of $\tilde{B}$ by $\xi_k$ and the two-by-two diagonal blocks $\tilde{B}_{kk}$ by

$$\tilde{B}_{kk} := \begin{pmatrix} u_k & x_k \\ v_k & w_k \end{pmatrix}, \quad u_k + w_k > 2\sqrt{u_k w_k - x_k v_k}\ , \ u_k w_k - x_k v_k > 0 \tag{4.4}$$

(the conditions on the entries of $\tilde{B}_{kk}$ ensure that its eigenvalues are distinct and positive, so that $\tilde{B}$ is diagonalizable). We shall require that $\{B, C\}$ generates an L-convergent iteration method. This requirement is crucial in order to quickly remove stiff components from the iteration error (see [7]). In fact, L-convergence implies that $Z^\nu(\infty)$ vanishes for $\nu \geq 2$, because the matrix $\tilde{Z}(z)$ is block-diagonal. Hence, within two inner iterations, all stiff error components are removed from the iteration error.
The following result provides a lower bound for $\rho(z)$ in the left halfplane when we impose the condition of L-convergence. The proof parallels the proof of a similar theorem in [9].

**Theorem 4.2.** Let $\{B, C\} = \{Q\tilde{B}Q^{-1}, Q\tilde{C}Q^{-1}\}$, let $\tilde{B}_{kk}$ be defined by (4.4), and let the generated PILSRK method be L-convergent. Then, we have in the left halfplane for all $a_k$ and $b_k$ the inequality

$$\rho := \max_{\mathrm{Re}(z) \leq 0} \rho(z) \geq 1 - \frac{\xi_\sigma}{\alpha_\sigma} \quad .$$

**Proof.** The eigenvalues $\mu_k(z)$ of the matrix $Z(z) = Q\tilde{Z}(z)Q^{-1}$ are given by those of the two-by-two diagonal blocks

$$\tilde{Z}_{kk}(z) := z\left(I - z\tilde{B}_{kk}\right)^{-1}\left(\tilde{A}_{kk} - \tilde{B}_{kk}\right), \ \eta_k \neq 0. \tag{4.5}$$

These eigenvalues satisfy the characteristic equation

$$(4.6) \qquad \det\begin{pmatrix} a_k - u_k - (z^{-1} - u_k)\mu_k(z) & b_k - x_k + x_k\mu_k(z) \\ c_k - v_k + v_k\mu_k(z) & 2\xi_k - a_k - w_k - (z^{-1} - w_k)\mu_k(z) \end{pmatrix} = 0.$$

It is easily verified that $\mu_k(z)$ vanishes at infinity if

$$(4.7) \qquad v_k = \frac{(a_k - 2\xi_k)u_k^2 + (2\alpha_k^2 + c_k x_k)u_k - a_k\alpha_k^2}{a_k x_k - b_k u_k}, \quad w_k = \frac{\alpha_k^2 + x_k v_k}{u_k},$$

where $x_k$ and $u_k$ are still free. In addition, we have to satisfy the inequalities in (4.4). Since $u_k w_k = \alpha_k^2 + x_k c_k$, these inequalities are satisfied if $u_k + w_k > 2\alpha_k$. Equation (4.6) is solved by

$$(4.8) \qquad \mu_k(z) = 0, \quad \mu_k(z) = \frac{(2\xi_k - u_k - w_k)z}{\alpha_k^2 z^2 - (u_k + w_k)z + 1}.$$

Since the function $\mu_k(z)$ is regular in the left half plane, its maximum in the left halfplane is assumed on the imaginary axis. It is easily verified that

$$(4.9) \qquad |\mu_k(iy)| = \frac{|2\xi_k - u_k - w_k|\,|y|}{\sqrt{(1 - \alpha_k^2 y^2)^2 + (u_k + w_k)^2 y^2}},$$

assumes an absolute maximum at $y = \pm\alpha_k^{-1}$ which is given by

$$(4.10) \qquad \max_{\mathrm{Re}(z)\le 0} \rho(\tilde{Z}_{kk}(z)) = \left| 1 - \frac{2\xi_k}{u_k + w_k} \right|, \quad u_k + w_k > 2\alpha_k.$$

This value is bounded below by $1 - \xi_k\alpha_k^{-1}$ (we recall that we have assumed $\xi_k > 0$). Hence, we see that the eigenvalues of A with the smallest ratio $\xi_k\alpha_k^{-1}$, that is, the eigenvalues with the relatively largest imaginary part, determine a lower bound for $\rho(z)$. Recalling that the ordering of the diagonal blocks $\tilde{A}_{kk}$ is such that $|\eta_k / \xi_k|$ increases with k, we conclude that in the left halfplane the maximal value of $\rho(z)$ is bounded below by $|1 - \xi_\sigma\alpha_\sigma^{-1}|$, proving the assertion of the theorem. ♦

From (4.10) it is clear that the best we can do is to choose $u_k + w_k = 2\theta_k\alpha_k$, where $\theta_k = 1 + \varepsilon_k$ with $\varepsilon_k$ a small positive number. By virtue of (4.7) we obtain the relation

$$u_k^2 + \alpha_k^2 + x_k\frac{(a_k - 2\xi_k)u_k^2 + (2\alpha_k^2 + c_k x_k)u_k - a_k\alpha_k^2}{a_k x_k - b_k u_k} = 2\theta_k\alpha_k u_k.$$

This equation shows that by choosing $x_k = 0$, we can compute $u_k$ *independently* of the values $a_k$, $b_k$ and $c_k$. Hence, we preserve a maximal amount of freedom in selecting the block-triangularizing matrix Q. Setting $x_k = 0$, we obtain

$$(4.11) \qquad u_k = \gamma_k\alpha_k, \; v_k = c_k\alpha_k\frac{(\gamma_k^2 - 1)a_k - 2\gamma_k^2\xi_k + 2\gamma_k\alpha_k}{\gamma_k(a_k^2 - 2\xi_k a_k + \alpha_k^2)}, \; w_k = \frac{\alpha_k}{\gamma_k}, \; \gamma_k := \theta_k \pm \sqrt{\theta_k^2 - 1},$$

where $\theta_k > 1$. By virtue of (4.10), we have the result:

$$(4.10') \qquad \max_{\mathrm{Re}(z)\le 0} \rho(\tilde{Z}_{kk}(z)) = \rho_k := 1 - \frac{2\gamma_k\xi_k}{(\gamma_k^2 + 1)\alpha_k} \quad , \quad \gamma_k \neq 1, \gamma_k > 0.$$

Furthermore, it follows from (4.8) that $Z(z)$ has a complete eigensystem for all finite $z$ (this is also true for $z = 0$, because $Z(0) = O$), and $\rho(\tilde{Z}_{kk}(z)) < 1$ in the region

$$(4.12) \qquad (2\gamma_k\xi_k - \gamma_k^2\alpha_k - \alpha_k)^2 |z|^2 < |\gamma_k\alpha_k^2 z^2 - (\gamma_k^2\alpha_k + \alpha_k)z + \gamma_k|^2.$$

Thus, we have proved:

**Theorem 4.3.** For $\gamma_k \neq 1, \gamma_k > 0$ and all $a_k$ and $b_k$, we have

(i) the matrix pair $\{B, C\} = \{Q\tilde{B}Q^{-1}, Q\tilde{C}Q^{-1}\}$ is in $\mathbb{B}(A)$ if $\tilde{B}$ satisfies (4.11),

(ii) the convergence region $\mathbb{\Gamma}$ is given by the cross section of the regions (4.12),

(iii) $\max_{\mathrm{Re}(z)\le 0} \rho(z) = \max_k \rho_k, \quad \rho_k := 1 - \frac{2\gamma_k\xi_k}{(\gamma_k^2 + 1)\alpha_k}$ . ◆

A comparison with Theorem 4.2 shows that values of $\gamma_k$ close to 1 yield an 'almost' minimal spectral radius. In Table 4.1, we have listed for a few Radau IIA methods the values of $\rho_k$ in the case $\gamma_k = 7/8$.

**Table 4.1.** Values of $\rho_k$ for s-stage Radau IIA methods ($\gamma_k = 7/8$).

| k | s = 2 | s = 4 | s = 6 | s = 8 |
|---|---|---|---|---|
| 1 | 0.19 | 0.06 | 0.03 | 0.02 |
| 2 | | 0.45 | 0.21 | 0.12 |
| 3 | | | 0.57 | 0.33 |
| 4 | | | | 0.64 |

These values are worst-case values, because in the greater part of the left halfplane, $\rho(z)$ is much smaller.

### 4.2. The averaged amplification factors

First we compute the averaged amplification factor for the eigenvector components of the iteration error associated with (3.1) after $\nu$ iterations. These components are amplified by $\tilde{Z}_{kk}^{\nu}(z)$, where $z = h\lambda$ corresponds with the (generalized) eigenvectors of $\{K_n, J_n\}$. Hence, the averaged amplification factor associated with (3.1) is defined by

$$\tilde{\rho}^{(\nu)}(z) := \max_k \tilde{\rho}_k^{(\nu)}(z), \quad \tilde{\rho}_k^{(\nu)}(z) := \sqrt[\nu]{\| \tilde{Z}_{kk}^{\nu}(z) \|} \ .$$

Here, $\tilde{\rho}_k^{(\nu)}(z)$ may be considered as the averaged amplification factor associated with the kth subsystem in (3.1). Let us set $\gamma_k = a_k\alpha_k^{-1}$, so that the matrix $\tilde{Z}_{kk}^{\nu}(z)$ simplifies to (cf. [9])

$$(4.13)\qquad \widetilde{Z}_{kk}{}^{\nu}(z) = \mu_k{}^{\nu}(z)\begin{pmatrix} 0 & q_k(z) \\ 0 & 1 \end{pmatrix},\ \mu_k(z) := \frac{b_k c_k z}{(1 - a_k z)(a_k - \alpha_k{}^2 z)},\ q_k(z) := \frac{a_k - \alpha_k{}^2 z}{c_k}.$$

With respect to the Euclidean norm, we have

$$\widetilde{\rho}_k{}^{(\nu)}(z) = |\mu_k(z)|\left(1 + |q_k(z)|^2\right)^{1/2\nu} = |\mu_k(z)|^{1-1/\nu}\left(|\mu_k(z)|^2 + |\mu_k(z)q_k(z)|^2\right)^{1/2\nu}.$$

We majorize this expression in the left halfplane by using the maximum value of $|\mu_k(z)|$ and the maximum value of $|\mu_k(z)q_k(z)|$. From Theorem 4.3 it follows that $|\mu_k(z)| \le \rho_k$ and an elementary calculation yields $|\mu_k(z)|^2 + |\mu_k(z)q_k(z)|^2 \le \rho_k{}^2 + b_k{}^2 a_k{}^{-2}$. Thus,

$$(4.14)\qquad \widetilde{\rho}_k{}^{(\nu)}(z) \le \rho_k \sqrt[2\nu]{\beta_k}\ ,\quad \beta_k = 1 + \frac{b_k{}^2}{a_k{}^2 \rho_k{}^2}\ .$$

Expressing the upperbound (4.14) in terms of the parameters $a_k$, $b_k$ and $c_k$, we obtain for the amplification factors in the transformed and untransformed space the estimates given in the theorem:

**Theorem 4.4.** If $\gamma_k = a_k \alpha_k{}^{-1}$, then with respect to the Euclidean norm, the averaged amplification factors $\widetilde{\rho}_k{}^{(\nu)}(z)$ and $\rho^{(\nu)}(z)$ satisfy in the left halfplane the inequalities

$$(4.15)\qquad \widetilde{\rho}_k{}^{(\nu)}(z) \le \rho_k \sqrt[2\nu]{\beta_k}\ ,\quad \rho_k = 1 - \frac{2a_k\xi_k}{a_k{}^2 + \alpha_k{}^2}\ ,\quad \beta_k = 1 + \frac{(a_k{}^2 + \alpha_k{}^2)^2}{a_k{}^2 c_k{}^2}\ ,$$

$$\rho^{(\nu)}(z) := \sqrt[\nu]{\| Z^{\nu}(z)\|}\ \le \sqrt[\nu]{\kappa(Q)}\ \max_k \widetilde{\rho}_k{}^{(\nu)}(z),\ \kappa(Q) := \|Q\|\,\|Q^{-1}\|.\ \blacklozenge$$

### 4.3. The transformation matrix

Theorem 4.4 suggests the use of transformation matrices Q such that $a_k \approx \alpha_k$ (to achieve that $\rho_k$ is close to its minimal value) and $c_k{}^2 >> 1$. Such transformation matrices can be constructed, however, as expected, they are poorly conditioned, so that we have fast convergence in the transformed space, but not necessarily fast convergence in the untransformed space. Here, we shall restrict our considerations to the Gauss-Seidel case $C \ne O$ with *orthogonal* transformation matrices (the Jacobi case $\{B, C\} = \{Q\widetilde{B}Q^{-1}, O\}$ with non-orthogonal Q has been analysed in [9]).

In order to construct an orthogonal matrix Q, let R be an orthogonal transformation matrix such that $\underline{S} := R^{-1}AR$ is a real Schur form of A with two-by-two diagonal blocks given by

$$\underline{S}_{kk} = \xi_k \ \text{ if } \eta_k = 0,\ \ \underline{S}_{kk} = \begin{pmatrix} \underline{a}_k & \underline{b}_k \\ \underline{c}_k & 2\xi_k - \underline{a}_k \end{pmatrix},\ \ \underline{b}_k\underline{c}_k = -(\underline{a}_k{}^2 - 2\xi_k\underline{a}_k + \alpha_k{}^2) \ \text{ if } \eta_k \ne 0.$$

The values of $\underline{a}_k$ and $\underline{b}_k$ are completely determined by R (for the construction of R we refer to [11]). We now transform these diagonal blocks by a block-diagonal rotation matrix $\Delta = (\Delta_{jk})$ with

$$\Delta_{kk} = 1 \ \text{ if } \eta_k = 0,\ \ \Delta_{kk} = \begin{pmatrix} \cos(\phi_k) & -\sin(\phi_k) \\ \sin(\phi_k) & \cos(\phi_k) \end{pmatrix} \ \text{ if } \eta_k \ne 0.$$

Then, $Q = R\Delta$, $\kappa(Q) = 1$ and $\widetilde{A} = \Delta^{-1}R^{-1}AR\Delta$, where the diagonal blocks of $\widetilde{A}$ are given by (2.3) with

$$a_k = \underline{a}_k \cos^2(\phi_k) - (\underline{b}_k + \underline{c}_k)\cos(\phi_k)\sin(\phi_k) + (2\underline{\xi}_k - \underline{a}_k)\sin^2(\phi_k),$$
$$c_k = \underline{c}_k \cos^2(\phi_k) + 2(\underline{a}_k - \underline{\xi}_k)\cos(\phi_k)\sin(\phi_k) - \underline{b}_k \sin^2(\phi_k).$$

The parameter $\phi_k$ is chosen such that the spectral radius $\rho_k$ occurring in the upper bound (4.15) is minimized. This means that $a_k$ should be close to $\alpha_k$. In order to avoid defect matrices $\widetilde{B}_{kk}$, we should not allow $a_k = \alpha_k$. Let us impose the constraints $a_k \le 7\alpha_k/8$ and $a_k \ge 9\alpha_k/8$. We now determine $\phi_k$ such that $|a_k - \alpha_k|$ is minimized subject to these constraints. If more than one values of $\phi_k$ are found that minimizes $|a_k - \alpha_k|$, then we take the value that also minimizes $\beta_k$.
For the s-stage Radau IIA methods with s = 2, 4, 6 and 8, we found that there exists $\phi_k$-values such that $a_k = 7\alpha_k/8$ (i.e. $\gamma_k = 7/8$). The corresponding $\rho_k$-values are listed in Table 4.1. For s = 4 and s = 8, the corresponding $\beta_k$-values are respectively given by {3.3, 2.0} and {2.6, 2.8, 2.4, 2.4}.
Table 4.2 lists the *actual* left halfplane upper bounds for $\rho^{(\nu)}(z)$ (in brackets, we listed for $\widetilde{C} \neq O$ the theoretical upperbounds of (4.15), which are too pessimistic by only less than 15%). The amplification factors for $\widetilde{C} = O$ are taken from [9] and turn out to be considerably larger.

**Table 4.2.** Actual upper bounds for $\rho^{(\nu)}(z)$ for Radau IIA ($\gamma_k = 7/8$).

| $\nu$ | s = 4 | | s = 8 | |
|---|---|---|---|---|
| | Jacobi | Gauss-Seidel | Jacobi | Gauss-Seidel |
| 1 | 1.95 | 0.54 (0.63) | 3.51 | 0.84 (0.98) |
| 2 | 0.98 | 0.49 (0.53) | 1.88 | 0.73 (0.80) |
| 3 | 0.76 | 0.48 (0.50) | 1.30 | 0.70 (0.74) |
| 4 | 0.66 | 0.47 (0.49) | 1.19 | 0.68 (0.71) |
| ... | | ... | | ... |
| $\infty$ | 0.45 | 0.45 (0.45) | 0.64 | 0.64 (0.64) |

### 4.4. Effect of the predictor

The averaged amplification factor $\rho^{(\nu)}(z)$ defined in Theorem 4.4 does not take the amplification effect of the predictor formula for $\mathbf{Y}^{(0)}$ needed in (3.2) into account. This effect plays a role in the first outer iteration. Let $\mathbf{Y}^{(1,0)} = \mathbf{Y}^{(0)}$ with $\mathbf{Y}^{(0)}$ defined by

(4.17) either $\Phi\big((h^{-1}[B+C]^{-1}\otimes I)(\mathbf{Y}^{(0)} - (E_P\otimes I)\mathbf{Y}_n), \mathbf{Y}^{(0)}\big) = \mathbf{0}$ or $\mathbf{Y}^{(0)} = (E_P\otimes I)\mathbf{Y}_n$.

where $\mathbf{Y}_n$ is the stage vector computed in the preceding step and the matrix $E_P$ can be used to control the order of accuracy of $\mathbf{Y}^{(0)}$. The second predictor formula in (4.17) is an extrapolation formula based on the back values contained in the preceding stage vector $\mathbf{Y}_n$. The first predictor formula is obtained from (2.1) by replacing A with B+C and $\mathbf{W}_n$ with a linear combination of back values contained in $\mathbf{Y}_n$, and may be considered as general linear method (GLM). Since B and B+C are similar to the same diagonal matrix, the GLM formula can be solved by a diagonalized modified Newton process with the same LU decompositions as used in (3.2). Hence, only FBSs are needed. To start the Newton process, we used an order s-1 accurate extrapolation formula based on $\mathbf{Y}_n$.

We consider the predictor effect for *linear* problems and for back information vectors given by $\mathbf{W}_n = (E\otimes I)\mathbf{Y}_n$, where $E := \mathbf{e}\mathbf{e}_s^T$. Let P be a matrix which equals either B+C or O and define $N_P := I\otimes K_n - P\otimes hJ_n$. Then, we may write the residual function (2.1) and the predictors (4.17) as

$$\mathbf{R}_n(\mathbf{Y}) = (h^{-1}A^{-1}\otimes I)\big(N_A\mathbf{Y} - (E\otimes K_n)\mathbf{Y}_n\big), \quad N_P\mathbf{Y}^{(0)} = (E_P\otimes K_n)\mathbf{Y}_n.$$

From (4.1) and (2.2) it follows that

$$\begin{aligned}(4.18)\qquad \mathbf{Y}^{(1,\nu)} - \mathbf{Y}^{(1)} &= M^\nu\big(\mathbf{Y}^{(1,0)} - \mathbf{Y}^{(1)}\big) = M^\nu\big(\mathbf{Y}^{(0)} - \mathbf{Y}^{(1)}\big) = M^\nu N_A^{-1}(hA\otimes I)\mathbf{R}_n(\mathbf{Y}^{(0)})\\ &= M^\nu\big(\mathbf{Y}^{(0)} - N_A^{-1}(E\otimes K_n)\mathbf{Y}_n\big) = M^\nu\big(N_P^{-1}(E_P\otimes K_n) - N_A^{-1}(E\otimes K_n)\big)\mathbf{Y}_n.\end{aligned}$$

Taking into account the computational effort involved in applying the predictor formula, we are led to the following definition of the averaged amplification factor associated with (4.18):

$$(4.19)\qquad \rho_{pred}^{(\nu)}(z) := \sqrt[\nu]{\| z^{\nu-\theta}(z)Z^*(z)\|}\ , \quad Z^*(z) := (I - zP)^{-1}E_P - (I - zA)^{-1}E,$$

where $\theta = 1$ if $P = B+C$ and $\theta = 0$ if $P = O$.
A qth-order accurate predictor is obtained by defining $E_P$ according to

$$(4.20)\qquad E_P\mathbf{e} = \mathbf{e}, \quad E_PX = U - PV, \quad U := \Big(\tfrac{1}{j}\,c^j\Big), V := \big(c^{j-1}\big), X := \Big(\tfrac{1}{j}\,(c - e)^j\Big), \quad j = 1, \ldots, q.$$

There are various options for choosing $E_p$. For $P = O$, we have considered the case $E_P = \mathbf{e}\mathbf{e}_s^T$ (last step value (LSV) predictor) and the case where $E_P$ is defined according to (4.20) with $q = s-1$ (maximal order extrapolation (EPL)). For $P = B+C$, we defined $E_P$ according to (4.20) with $q = 2$ and we used the remaining free parameters to minimize both $\rho_{pred}^{(1)}(z) + \rho_{pred}^{(2)}(z)$ in the left half plane (GLM predictor). Table 4.3 lists left halfplane upper bounds for $\rho_{pred}^{(\nu)}(z)$ in the case of the 4-stage Radau IIA method. In appreciating these values, we should take the effect of the *order of accuracy* of the predictor into account. For example, the LSV predictor together with the Gauss-Seidel version $\widetilde{C} \neq O$ possesses the smallest left halfplane upper bounds for $\rho_{pred}^{(\nu)}(z)$, but its zero order will be a drawback (see Section 5).

**Table 4.3.** Actual upper bounds for $\rho_{pred}^{(\nu)}(z)$ for 4-stage Radau IIA with $\gamma_k = 7/8$.

| | Jacobi | | | Gauss-Seidel | | |
|---|---|---|---|---|---|---|
| ν | LSV (q=0) | EPL (q=3) | GLM (q=2) | LSV (q=0) | EPL (q=3) | GLM (q=2) |
| 1 | 2.66 | 6.98 | 3.48 | 0.92 | 39.5 | 2.93 |
| 2 | 1.15 | 2.27 | 1.30 | 0.61 | 4.17 | 0.88 |
| 3 | 0.84 | 1.34 | 0.91 | 0.54 | 1.97 | 0.67 |
| 4 | 0.71 | 1.02 | 0.76 | 0.51 | 1.36 | 0.61 |
| ... | ... | ... | ... | ... | ... | ... |
| ∞ | 0.45 | 0.45 | 0.45 | 0.45 | 0.45 | 0.45 |

### 4.5. Comparison of LU and FBS costs

We conclude this section by summarizing the total and effective LU costs per Jacobian update and the total and effective FBS costs per outer iteration. Table 4.4 lists these costs for the block-diagonalized Newton, the Newton-PILSRK method derived above, and Newton-PILSRK methods with C = O. The cost formulas are given for the cases s even and s odd, assuming that the matrix A has no real eigenvalues if s is even and only one real eigenvalue if s is odd. In the case of the PILSRK $\{\widetilde{Q}\widetilde{B}Q^{-1}, \widetilde{Q}\widetilde{C}Q^{-1}\}$ method, $\bar{r}$ and $\bar{r}_2$ respectively denote the averaged number of inner iterations over *all* subsystems and over subsystems associated with the *complex* eigenvalue pairs of A. Finally, r denotes the maximal number of inner iterations needed for the s subsystems in the PILSRK {B, O} method. The figures in Table 4.4 show that the two PILSRK methods require the same number of (total and effective) LU operations. Their effective FBS costs are highly dependent on the value of $\bar{r}$, $\bar{r}_2$ and r.

**Table 4.4.** Total and effective LU and FBS costs for even or odd numbers of RK stages.

| Method | Total LU / $d^3$ | Total FBS / $d^2$ | Eff. LU / $d^3$ | Eff. FBS / $d^2$ |
|---|---|---|---|---|
| Block-diagonalized Newton | $\frac{4s}{3}$ or $\frac{2}{3}(2s-1)$ | $4s$ or $4s-2$ | $\frac{8}{3}$ | 8 |
| PILSRK: Gauss-Seidel | $\frac{2s}{3}$ | $2s\bar{r}$ or $2(s\bar{r}_2 - \bar{r}_2 + 1)$ | $\frac{2}{3}$ | $s\bar{r}$ or $s\bar{r}_2 - \bar{r}_2 + 2$ |
| PILSRK: Jacobi | $\frac{2s}{3}$ | $2sr$ | $\frac{2}{3}$ | $2r$ |

### 5. Numerical experiments

The aim of this section is to compare the algorithmic properties of the Newton-PILSRK method {(2.2),(3.2)}. We compare the Gauss-Seidel version $\widetilde{C} \neq O$ with Q orthogonal as analysed in this paper and the Jacobi version $\widetilde{C} = O$ with Q nonorthogonal analysed in [9]. In both cases, we take $\gamma_k = 7/8$. The comparisons are carried out for a few test problems from the literature.

The corrector is defined by the 4-stage Radau IIA corrector and the predictor formula is either the LSV or the EPL predictor (see Section 4.4)), and is specified in the tables of results.

Diagonalizing (3.2) by the transformation $\mathbf{Y}^{(j)} = (S\otimes I)\mathbf{X}^{(j)}$ and writing $A^* = S^{-1}AS$, $B^* = S^{-1}BS$, ..., yields the method

$$(5.1)\qquad \left(I\otimes K_n - B^*\otimes hJ_n\right)\left(\mathbf{X}^{(j,v)} - \mathbf{X}^{(j,v-1)}\right) = -\left(I\otimes K_n - (A^* - C^*)\otimes hJ_n\right)\mathbf{X}^{(j,v-1)}$$
$$+ (C^*\otimes hJ_n)\mathbf{X}^{(j)} + \left(I\otimes K_n - A^*\otimes hJ_n\right)\mathbf{X}^{(j-1)} - (hS^{-1}A\otimes I)\mathbf{R}(\mathbf{Y}^{(j-1)}),$$

where $B^*$ is diagonal. In the Jacobi case, we have (cf. [9])

$$B = \begin{pmatrix} 0.1096 & -0.0430 & 0.0268 & -0.0080 \\ 0.2085 & 0.3064 & -0.0671 & 0.0211 \\ 0.2484 & 0.0823 & 0.2573 & -0.0142 \\ 0.2596 & -0.0515 & 0.4219 & 0.0780 \end{pmatrix}, \quad C = O,$$

$$S = \begin{pmatrix} 2.9526 & 0.3159 & 1.5325 & 0.0276 \\ -7.2663 & -0.8756 & -1.0553 & -0.3113 \\ 3.4202 & 0.9493 & -10.7997 & -2.1349 \\ 34.8970 & 4.3753 & -42.9039 & -5.8960 \end{pmatrix}, B^* = \begin{pmatrix} 0.1521 & 0 & 0 & 0 \\ 0 & 0.1986 & 0 & 0 \\ 0 & 0 & 0.1736 & 0 \\ 0 & 0 & 0 & 0.2269 \end{pmatrix}.$$

and in the Gauss-Seidel case

$$B = \begin{pmatrix} 0.1175 & -0.0207 & 0.0255 & -0.0017 \\ 0.2555 & 0.2758 & -0.0535 & 0.0037 \\ -0.0256 & -0.0076 & 0.2030 & -0.0002 \\ 0.0206 & 0.0528 & 0.3488 & 0.1549 \end{pmatrix}, C = \begin{pmatrix} -0.0061 & -0.0117 & 0.0002 & -0.0019 \\ -0.0281 & -0.0400 & -0.0002 & -0.0059 \\ 0.2492 & 0.3955 & -0.0010 & 0.0614 \\ 0.2099 & 0.3114 & 0.0007 & 0.0470 \end{pmatrix}$$

$$S = \begin{pmatrix} 0.2030 & 0.3803 & -0.0763 & -0.0904 \\ -0.9495 & -0.8908 & 0.1977 & 0.2080 \\ 0.0858 & 0.1003 & -0.1466 & -0.0182 \\ -0.2233 & -0.2275 & -0.9662 & -0.9738 \end{pmatrix}, B^* = \begin{pmatrix} 0.2269 & 0 & 0 & 0 \\ 0 & 0.1737 & 0 & 0 \\ 0 & 0 & 0.1986 & 0 \\ 0 & 0 & 0 & 0.1521 \end{pmatrix}.$$

Since this paper aims at a comparison of algorithmic properties, we avoided effects of stepsize and iteration strategies by performing the experiments with fixed stepsizes h and fixed numbers of outer iterations m and inner iterations r. Furthermore, the Jacobian and the LU-decompositions were computed in each integration step.

The tables of results list the minimal number of correct digits at the end point:

(5.2) $\quad cd := -\log_{10} \| \mathbf{y}_{end} - \mathbf{y}(t_{end}) \|_\infty .$

Here, $\mathbf{y}_{end}$ denotes the numerical solution at the end point $t_{end}$. Negative cd values will be denoted by cd = -.

### 5.1. The transistor amplifier (index 1)

The first test problem is the transistor amplifier given in [6] on the interval [0, 0.2] (see also [10]). This nonlinear, eight-dimensional problem of index 1 can be represented in the implicit form $K\mathbf{y}' = \mathbf{f}(t,\mathbf{y})$ with a constant, singular capacity matrix K. Table 5.1 lists results for the EPL predictor and $h = 2_{10}{}^{-4}$. In both versions, only two inner iteration are needed to produce the same accuracy as the modified Newton process, but taking just one inner iteration seems to be the most efficient strategy. Furthermore, in accordance with Table 4.3, the Jacobi version performs better than the Gauss-Seidel version (note that the outer iteration process converges relatively slowly).

**Table 5.1.** Transistor amplifier with EPL predictor and $h = 2_{10}{}^{-4}$

| | Jacobi version | | | Newton | | Gauss-Seidel version | |
|---|---|---|---|---|---|---|---|
| m | r = 1 | r = 2 | -> | r = ∞ | <- | r = 2 | r = 1 |
| 1 | - | 4.6 | -> | 4.6 | <- | 4.6 | - |
| 2 | 6.5 | 6.6 | -> | 6.6 | <- | 6.6 | - |
| 3 | 7.7 | 7.5 | -> | 7.5 | <- | 7.5 | 7.0 |
| 4 | 8.1 | 8.0 | -> | 8.0 | <- | 8.0 | 7.2 |
| ... | ... | ... | | ... | | ... | ... |
| ∞ | 9.7 | 9.7 | -> | 9.7 | <- | 9.7 | 9.7 |

Next, we apply the LSV predictor. According to Table 4.3, now Gauss-Seidel should be the superior one. Table 5.2 shows that Gauss-Seidel does perform slightly better than Jacobi. Furthermore, a comparison with Table 5.1 reveals that the EPL predictor is considerably more efficient than the LSV predictor because of its higher order. We also tested the GLM predictor, but it could not beat the EPL predictor. Apparently, a higher order of accuracy is more important than smaller amplification factors.

**Table 5.2.** Transistor amplifier with LSV predictor and $h = 2_{10}{}^{-4}$

| | Jacobi version | | | | | Newton | | Gauss-Seidel version | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | r = 1 | r = 2 | r = 3 | r = 4 | -> | r = ∞ | <- | r = 4 | r = 3 | r = 2 | r = 1 |
| 1 | - | 2.1 | 2.9 | 3.1 | -> | 3.2 | <- | 3.1 | 2.8 | 2.0 | 1.1 |
| 2 | 1.4 | 3.7 | 4.7 | 4.4 | -> | 4.4 | <- | 4.4 | 4.2 | 3.7 | 2.0 |
| 3 | 2.5 | 4.9 | 5.9 | 5.8 | -> | 5.8 | <- | 5.8 | 5.6 | 4.9 | 2.7 |
| 4 | 3.4 | 6.0 | 6.6 | 6.7 | -> | 6.7 | <- | 6.8 | 6.9 | 6.2 | 3.7 |
| ... | ... | ... | ... | ... | | ... | | ... | ... | ... | ... |
| ∞ | 9.7 | 9.7 | 9.7 | 9.7 | -> | 9.7 | <- | 9.7 | 9.7 | 9.7 | 9.7 |

### 5.2. The car axis problem (index 3)

Table 5.2 presents results for the more complicated index 3 car axis problem consisting of 10 DAEs [10]. As in Table 5.1, Jacobi is slightly better than Gauss-Seidel and a one-inner-iteration strategy is most efficient (note that here the outer iteration process converges relatively fast).

**Table 5.3.** The car axis problem with EPL predictor and $h = 3_{10}{}^{-3}$

| | Jacobi version | | | | | Newton | | Gauss-Seidel version | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| m | r = 1 | r = 2 | r = 3 | r = 4 | -> | r = ∞ | <- | r = 4 | r = 3 | r = 2 | r = 1 |
| 1 | - | - | 0.8 | 2.5 | -> | 2.5 | <- | 2.5 | 0.8 | - | - |
| 2 | 1.8 | 2.0 | 2.4 | 5.4 | -> | 5.4 | <- | 5.4 | 2.4 | 1.9 | - |
| 3 | 1.9 | 5.3 | 6.5 | 6.5 | -> | 6.6 | <- | 6.6 | 5.4 | 4.3 | 1.8 |
| 4 | 2.4 | 6.3 | 6.6 | 6.6 | -> | 6.6 | <- | 6.6 | 6.6 | 6.0 | 2.6 |
| ... | ... | ... | ... | ... | | ... | | ... | ... | ... | ... |
| ∞ | 6.6 | 6.6 | 6.6 | 6.6 | -> | 6.6 | <- | 6.6 | 6.6 | 6.6 | 6.6 |

### 5.3. Concluding remarks

From the results in the Tables 5.1, 5.2 and 5.3 we may draw the following conclusions:

(i) The PILSRK inner iteration process profits most from high-order predictors.

(ii) If higher-order predictors like EPL are used, then the Gauss-Seidel version $\widetilde{C} \neq O$ converges slightly slower than the Jacobi version $\widetilde{C} = O$.

(iii) If the number of outer iterations m increases, then the number of inner iterations r can be chosen smaller.

(iv) In a (fixed m, fixed r) strategy, the one-inner-iteration strategy together with a high-order predictor seems to be most efficient. A dynamic iteration strategy is expected to perform several inner iterations in the first few outer iterations and only one inner iteration in the later outer iterations.

**References**

[1] Bendtsen, C. (1996): Highly stable parallel Runge-Kutta methods, Appl. Numer. Math. 21, 1-8.

[2] Brenan, K.E., Campbell, S.L. & Petzold, L.R. (1989): Numerical solution of initial-value problems in differential-algebraic equations, North-Holland, New York.

[3] Butcher, J.C. (1976): On the implementation of implicit Runge-Kutta methods, BIT 16, 237-240.

[4] Hairer, E. & Wanner, G. (1996): Solving ordinary differential equations, II. Stiff and differential-algebraic problems, Springer-Verlag, Berlin.

[5] Hairer, E. (1996): RADAUP, available via WWW at URL:
ftp: //ftp.unige.ch/pub/doc/math/stiff/radaup.f.

[6] Hairer, E., Lubich, Chr, & Roche, M. (1989): The numerical solution of differential-algebraic systems by Runge-Kutta methods, LNM 1409, Springer, Berlin.

[7] Houwen, P.J. van der, & Sommeijer, B.P. (1991): Iterated Runge-Kutta methods on parallel computers, SIAM J. Sci. Stat. Comput. 12, 1000-1028.

[8] Houwen, P.J. van der, & Swart, J.J.B. de (1995): Triangularly implicit iteration methods for ODE-IVP solvers, Preprint NM-R9510, CWI, Amsterdam (to appear in SIAM J. Sci. Comput.).

[9] Houwen, P.J. van der, & Swart, J.J.B. de (1996): Parallel linear system solvers for Runge-Kutta methods, Preprint NM-R9616, CWI, Amsterdam (to appear in Advances in Computational Mathematics).

[10] Lioen, W.M., Swart, J.J.B. de & Veen, W.A. van der (1996): Test set for IVP solvers, available via WWW at URL:
http: //www.cwi.nl/cwi/projects/IVPtestset.shtml.

[11] Messina, E., Swart, J.J.B. de & Veen, W.A. van der (1996): Parallel iterative linear solvers for Multistep Runge-Kutta methods, Preprint NM-R9619, CWI, Amsterdam.

[12] Orel, B. [1993]: Parallel Runge-Kutta methods with real eigenvalues, Appl. Numer. Math. 11, 241-250.

# Chapter 8

# Waveform Relaxation Methods for Implicit Differential Equations

P.J. van der Houwen and W.A. van der Veen

**Abstract**

We apply a Runge-Kutta-based waveform relaxation method to initial-value problems for implicit differential equations. In the implementation of such methods, a sequence of nonlinear systems has to be solved iteratively in each step of the integration process. The size of these systems increases linearly with the number of stages of the underlying Runge-Kutta method, resulting in high linear algebra costs in the iterative process for high-order Runge-Kutta methods. In our earlier investigations of iterative solvers for implicit initial-value problems, we designed an iteration method in which the linear algebra costs are almost independent of the number of stages when implemented on a parallel computer system. In this paper, we use this parallel iteration process in the Runge-Kutta waveform relaxation method. In particular, we analyse the convergence of the method. The theoretical results are illustrated by a few numerical examples.

*CR Subject Classification (1991):* G1.7
*Keywords & Phrases:* numerical analysis, implicit differential equations, convergence, waveform relaxation, Runge-Kutta methods, parallelism.
*Note:* The research reported in this paper was partly supported by the Technology Foundation (STW) in the Netherlands.

## 1 Introduction

Consider the initial-value problem (IVP) for the implicit differential equation (IDE)

$$\phi(t, y', y) = 0, \quad t_0 \le t \le t_{\text{end}}, \quad y,\ \phi \in R^d. \tag{1}$$

It will be assumed that the initial conditions for $y(t_0)$ and $y'(t_0)$ are consistent and that the IVP has a unique solution. Furthermore, defining the Jacobian matrices $K := \phi_u(t,u,v)$ and $J := -\phi_v(t,u,v)$, it will be assumed that in the neighbourhood of the solution, the characteristic equation $\det(\lambda K - J) = 0$ associated with the linearization of (1) has only zeros in the nonpositive halfplane. The pair of matrices $\{K,J\}$ will be said to be a *stable pair* if they satisfy this requirement. In the convergence analysis of iteration methods for solving the numerical discretization of (1), the property of matrix pairs will play a central role.

A large class of numerical discretizations of (1) is defined by

$$\begin{aligned} &y_n = (e_s^T \otimes I)Y_n, \\ &\Phi(et_{n-1} + ch, (h^{-1}A^{-1} \otimes I)(Y_n - (E \otimes I)Y_{n-1}), Y_n) = 0. \end{aligned} \tag{2}$$

Here, $y_n$ is the numerical approximation to the exact solution value $y(t_n)$, $A$ and $E$ denote $s \times s$ matrices, $e_s$ is the $s$th unit vector, $h$ is the step size $t_n - t_{n-1}$ (to be assumed constant in the analysis presented in this paper), $\otimes$ denotes the Kronecker product, and $I$ is the $d \times d$ identity matrix (in the following, we shall use the notation $I$ for any identity matrix; its dimension will always be clear from the context). The $s$ components $Y_{ni}$ of the $sd$-dimensional stage vector $Y_n$ represent approximations to the exact solution values $y(t_{n-1} + c_i h)$, where the $c_i$ are the components of the abscissa vector $c = (c_i)$ and where $c_s = 1$. Furthermore, for any pair of vectors $Y_n' = (Y_{ni}')$ and $Y_n = (Y_{ni})$, we define the function

$$\Phi(et + ch, Y_n', Y_n) := (\phi(t + c_i h, Y_{ni}', Y_{ni})). \tag{3}$$

The method (2) is completely defined by the triple $\{A, E, c\}$. We remark that (2) reduces to a (stiffly accurate) RK method for IDEs if $A$ equals the Butcher matrix of the RK method, $c := Ae$, and $E := (0, \ldots, 0, e)$, $e$ being the $s$-dimensional vector with unit entries (see [4]).

In [8], parallel iteration methods for solving the stage vector $Y_n$ from the nonlinear system (2) have been proposed. In this paper, we want to combine these parallel iteration techniques with the waveform relaxation (WR) approach. The resulting numerical solution methods have a considerable amount of intrinsic parallelism. However, the price to be paid is a decrease of the speed of convergence of the iteration methods. This paper studies how the convergence of the WR method is influenced by the number of WR iterations, the number of modified Newton iterations, and the number of inner iterations (for solving the linear Newton systems). The theoretical results are illustrated by a few numerical examples.

## 2 WR methods

The derivation of WR methods starts with representing the IDE (1) in the form

$$\psi(t, y', y', y, y) = 0, \quad t_0 \le t \le t_{\text{end}}, \quad y,\ \psi \in R^d, \tag{4}$$

where $\psi(t, u', v', u, v)$ is a *splitting function* satisfying

$$\psi(t, u', u', u, u) = \phi(t, u', u).$$

This splitting function is chosen such that the Jacobian matrices $K^* := \partial\psi/\partial u'$ and $J^* := -\partial\psi/\partial u$ have a simple structure, so that, given an approximation $y^{(k-1)}$ to the solution $y$ of (1), a next approximation $y^{(k)}$ is more easily solved from the system

$$\psi(t, y'^{(k)}, y'^{(k-1)}, y^{(k)}, y^{(k-1)}) = 0, \ t_0 \le t \le t_{\text{end}}, \ y^{(k)},\ y^{(k-1)},\ \psi \in R^d, \tag{5}$$

than $y$ is solved from (1). Here, $k = 1, 2, ..., q$, and $y^{(0)}$ denotes an initial approximation to the solution of (1). The iteration process (5) is called *continuous WR iteration* with WR iterates $y^{(k)}$. Such iteration processes were introduced in Lelarasmee [9] and Lelarasmee, Ruehli & Sangiovanni-Vincentelli [10]. For linear problems, its convergence has been extensively studied in [12].

In the case of *explicit* differential equations (i.e., $K = K^* = I$), a popular choice for the splitting function $\psi$ is such that the matrix $J^*$ is $\sigma \times \sigma$ block-diagonal (block-Jacobi WR method). Then, each iteration of the WR method (5) requires the integration of $\sigma$ uncoupled systems over the interval $[t_0, t_{\text{end}}]$ (note that these integrations can be done in parallel on $\sigma$ processors). In the IDE case ($K \neq I$), we obtain a block-Jacobi WR method if both $J^*$ and $K^*$ are $\sigma \times \sigma$ block-diagonal. As an example, we consider the case where (5) is of the form

$$\psi_1(t, u'^{(k)}, y'^{(k-1)}, u^{(k)}, y^{(k-1)}) = 0, \quad t_0 \le t \le t_{\text{end}}, \quad u^{(k)}, \psi_1 \in R^{d_1},$$

$$\psi_2(t, v'^{(k)}, y'^{(k-1)}, v^{(k)}, y^{(k-1)}) = 0, \quad t_0 \le t \le t_{\text{end}}, \quad v^{(k)}, \psi_2 \in R^{d_2}.$$

Here, $d_1 + d_2 = d$ and $y = (u^T, v^T)^T$. Obviously, $K^*$ and $J^*$ are both $2 \times 2$ block-diagonal. More generally, whenever $K^*$ and $J^*$ are both $\sigma \times \sigma$ block-diagonal, we find a set of $\sigma$ subsystems with the generic form

$$\psi(t, y'^{(k)}, x'^{(k-1)}, y^{(k)}, x^{(k-1)}) = 0, \quad t_0 \le t \le t_{\text{end}}, \tag{6}$$

where $x^{(k-1)}$ is defined by the $\sigma$ subsystems solutions of the preceding WR iteration and $y^{(k)}$ is the new subsystem solution. For further details we refer to ([2], p. 276 ff.).

The convergence of the continuous WR iteration (5) is faster as the integration interval $[t_0, t_{\text{end}}]$ is smaller. In fact, for ordinary differential equations (ODEs) which arise for $K = I$, and for sufficiently smooth splitting functions $\psi$, we have the well-known estimate

$$\| y^{(k)}(t) - y(t) \| \leq \frac{L^k (t - t_0)^k}{k!} \max_{t_0 \leq \tau \leq t_{\text{end}}} \| y^{(0)}(\tau) - y(\tau) \|,$$

where $L$ is a constant depending on the splitting function (for example, for the standard test equation defined by $\phi = y' - \lambda y$ with splitting function $\psi = u' - \lambda v$, we have $L = |\lambda|$). This estimate indicates that convergence is improved if $t_{\text{end}} - t_0$ is small. Therefore, we do not apply the WR method on the whole interval $[t_0, t_{\text{end}}]$, but successively on a number of smaller subintervals (also called *windows*) of length $\omega h$ where $\omega$ is a usually small integer and $h$ the step size.

## 2.1 Discrete WR iteration

Let us integrate the IVP for (5) numerically by the step-by-step method $\{A, E, c\}$ defined in (2). Introducing the residual function

$$\begin{aligned} R(U, V, X) \quad := \quad & \Psi(et_{n-1} + ch, (h^{-1}A^{-1} \otimes I)(U - (E \otimes I)V), \\ & (h^{-1}A^{-1} \otimes I)(X - (E \otimes I)V), U, X), \end{aligned} \tag{7}$$

and dividing $[t_0, t_{\text{end}}]$ into subintervals (or windows) $[t_{\kappa\omega}, t_{\kappa\omega+\omega}]$, we obtain on $[t_{\kappa\omega}, t_{\kappa\omega+\omega}]$ the scheme

$$\begin{aligned} &\texttt{for} \quad k = 1 \quad \texttt{to} \quad q \\ &\qquad Y_{\kappa\omega}^{(k)} := Y_{\kappa\omega}^{(q)} \\ &\qquad \texttt{for} \quad n = \kappa\omega + 1 \quad \texttt{to} \quad \kappa\omega + \omega \\ &\qquad\quad \texttt{solve} \quad Y_n^{(k)} \quad \texttt{from} \quad R(Y_n^{(k)}, Y_{n-1}^{(k)}, Y_n^{(k-1)}) = 0, \\ &\qquad\quad \texttt{set} \quad y_n^{(k)} = (e_s^T \otimes I) Y_n^{(k)}. \end{aligned} \tag{8}$$

Here, $y_n^{(k)}, Y_n^{(k)}$, and $\Psi$ are the analogues of $y_n, Y_n$ and $\Phi$ occurring in (2). The scheme (7) - (8) will be called the *discrete WR iteration process* with (discrete) WR iterates $Y_n^{(k)}$ and $y_n^{(k)}$.

If (7) - (8) converges on all windows as $q \to \infty$, then $Y_n^{(q)}$ converges to the solution $Y_n$ of $R(Y_n, Y_{n-1}, Y_n) = 0$, that is, to the stage vector $Y_n$ defined in (2). As a consequence, $(e_s^T \otimes I) Y_n^{(q)}$ approximates the solution of (1) at $t_n$ with order $p$ in $h$, $p$ being the order of accuracy of the underlying method (2).

The iteration scheme (7) - (8) has a certain amount of intrinsic parallelism, because for a given subinterval $[t_{\kappa\omega}, t_{\kappa\omega+\omega}]$ and given $k$, the $\omega$ iterates $\{Y_{\kappa\omega+1}^{(k)},$

$Y_{\kappa\omega+2}^{(k-1)}, \ldots, Y_{\kappa\omega+\omega}^{(k+1-\omega)}\}$ can be computed in parallel (parallelism across the steps within a window, see e.g., [14] and [1]). Hence, effectively, the subinterval $[t_{\kappa\omega}, t_{\kappa\omega+\omega}]$ does not require the computation of $q\omega$ iterates, but only $q+\omega-1$ iterates, so that the number of *effective* (or *sequential*) WR iterations per step is $1+\omega^{-1}(q-1)$. Here, each iterate has dimension $sd$. Note that this holds for any splitting function $\psi$.

There is an additional amount of intrinsic parallelism if the splitting function $\psi$ is such that $J^*$ and $K^*$ are $\sigma \times \sigma$ block-diagonal. In such cases, the IVP can be decoupled into a set of $\sigma$ subsystems of the form (6) each of which can be integrated by the method $\{A, E, c\}$ defined in (2). Since these integrations can be done concurrently, the strategy described above can be applied to each subsystem. Thus, the effective costs per step reduce to the computation of $1+\omega^{-1}(q-1)$ WR iterates of dimension $sd^*$, where $d^*$ is the maximal dimension of the subsystems.

## 2.2 The Newton iteration process

In an actual application of (7) - (8), each time step requires the solution of $Y_n^{(k)}$ from the (nonlinear) system $R(Y_n^{(k)}, Y_{n-1}^{(k)}, Y_n^{(k-1)}) = 0$. Given the WR iteration index $k$ and the time step index $n$, we shall use the following iteration process:

$$
\begin{aligned}
&Y_n^{(k,0)} := Y_n^{(k-1,m)},\\
&\texttt{for}\quad j=1\quad \texttt{to}\quad m \qquad\qquad (9)\\
&\quad \texttt{solve}\quad Y_n^{(k,j)}\quad \texttt{from}\\
&\qquad N_0(Y_n^{(k,j)} - Y_n^{(k,j-1)}) = -h(A\otimes I)R(Y_n^{(k,j-1)}, Y_{n-1}^{(k)}, Y_n^{(k-1,m)}),
\end{aligned}
$$

where $N_0$ is the (modified) Newton matrix

$$N_0 := I \otimes K^* - A \otimes hJ^*. \qquad (10)$$

Here, the Jacobian matrices $K^*$ and $J^*$ of the splitting function $\psi$ are both evaluated at the step point $t_{n-1}$. The modified Newton process (9), will be assumed to be convergent.

The combination of the WR iteration method (7) - (8) and the modified Newton method (9) - (10) is a nested iteration process containing four loops with indices $\kappa$, $k$, $n$ and $j$. The three iteration parameters $q$, $\omega$, and $m$ determine the range of the indices $k$, $n$ and $j$. The number of effective modified Newton iterations (i.e., linear system solves) per step in $\{(7), (8), (9)\}$ is given by $m(1+\omega^{-1}(q-1))$.

**Remark 2.1.** In practice, it may be an efficient strategy to perform only a few Newton iterations, because the WR iterate $Y_n^{(k)}$ may still be far away from the solution $Y_n$ of (2). Hence, it seems a waist to perform many Newton

iterations for computing a close approximation to $Y_n^{(k)}$, which itself is a poor approximation to $Y_n$. In the extreme case where $m = 1$, the method $\{(7), (8), (9)\}$ reduces to

$$\begin{array}{l} \texttt{for} \quad k = 1 \quad \texttt{to} \quad q \\ \qquad Y_{\kappa\omega}^{(k)} := Y_{\kappa\omega}^{(q)} \\ \qquad \texttt{for} \quad n = \kappa\omega + 1 \quad \texttt{to} \quad \kappa\omega + \omega \\ \qquad\quad \texttt{solve} \quad Y_n^{(k)} \quad \texttt{from} \\ \qquad\qquad N_0(Y_n^{(k)} - Y_n^{(k-1)}) = -h(A \otimes I)R(Y_n^{(k-1)}, Y_{n-1}^{(k)}, Y_n^{(k-1)}) \\ \qquad\quad \texttt{set} \quad y_n^{(k)} = (e_s^T \otimes I)Y_n^{(k)}. \end{array} \tag{11}$$

A comparison with $\{(7), (8), (9)\}$ shows that in (11) we have a more frequent updating of the righthand side, so that it is expected that (11) shows a better overall convergence than $\{(7), (8), (9)\}$ with $m > 1$, that is, for constant $qm$, the accuracy is expected to be best for $m = 1$. However, it should also be observed that small $m$ implies more frequent communication when implemented on a parallel computer system, so that given the number of WR iterations $q$, the effective costs for $m = 1$ and $m = 2$ or $m = 3$ may well be comparable.

Let us consider the case where the matrices $K^*$ and $J^*$ are $\sigma \times \sigma$ lower block-triangular matrices $(K_{ij}^*)$ and $(J_{ij}^*)$. In order to see the amount of parallelism inherent to the resulting modified Newton matrix we reorder the rows and columns in $N_0$. Let the partitioning of the vector $y$ in (1) corresponding to the blocks $(K_{ij}^*)$ and $(J_{ij}^*)$ be denoted by $y = (u^T, v^T, \ldots)^T$, and let us replace the $sd$-dimensional vectors $Y$ in (9) by permuted vectors

$$\tilde{Y} = PY := (U^T, V^T, \ldots)^T,$$

where $P$ is such that $U$, $V$, ... are stage vectors associated with $u$, $v$, ... in the same way as $Y$ is associated with $y$. Then the permuted version of the linear system in (9) becomes

$$\begin{aligned} &\tilde{N}_0(\tilde{Y}_n^{(k,j)} - \tilde{Y}_n^{(k,j-1)}) = -P(hA \otimes I)R(Y_n^{(k,j-1)}, Y_{n-1}^{(k)}, Y_n^{(k-1,m)}), \\ &\tilde{N}_0 := PN_0P^{-1}. \end{aligned} \tag{12}$$

It is easily verified that for any matrix $C$ and any $\sigma \times \sigma$ block matrix $M = (M_{ij})$, the matrix $P(C \otimes M)P^{-1}$ becomes a $\sigma \times \sigma$ block matrix with entries $C \otimes M_{ij}$. Hence,

$$\begin{aligned} \tilde{N}_0 &:= (I \otimes K_{ij}^*) - (A \otimes hJ_{ij}^*), \\ &= \begin{pmatrix} I \otimes K_{11}^* - A \otimes hJ_{11}^* & O & \cdots \\ I \otimes K_{21}^* - A \otimes hJ_{21}^* & I \otimes K_{22}^* - A \otimes hJ_{22}^* & \cdots \\ \vdots & \vdots & \vdots \end{pmatrix}. \end{aligned} \tag{13}$$

This expression shows that solving (9) by a direct method requires the LU decomposition of the $\sigma$ diagonal blocks $I \otimes K^*_{ii} - A \otimes hJ^*_{ii}$. Hence, there are $\sigma$ LU decompositions to be performed which can all be done in parallel. The maximal dimension of the matrices to be decomposed equals $sd^*$, $d^*$ denoting the dimension of the largest blocks in $K^*$ and $J^*$, so that the effective LU costs on $\sigma$ processors is $O((sd^*)^3)$, each time the matrix $N_0$ in (9) is updated. Apart from these LU costs, each modified Newton iteration requires the evaluation of the function $R$ and a forward/backward substitution. The evaluation of $R$ can again be distributed over $\sigma$ processors.

## 2.3 Iterative solution of the Newton systems

The LU decompositions needed in the modified Newton process may be costly if $d^*$ is still large. Therefore, the linear Newton systems in (9) will be solved iteratively by an *inner* iteration process (in this connection, we may interpret the Newton process (9) - (10) as an *outer* iteration process). We shall use the iteration method

$$\begin{aligned} &U^{(0)} := Y_n^{(k,j-1)} \\ &C^{(k,j)} := N_0 Y_n^{(k,j-1)} - h(A \otimes I)R(Y_n^{(k,j-1)}, Y_{n-1}^{(k)}, Y_n^{(k-1,m)}) \\ &\texttt{for} \quad \nu = 1 \quad \texttt{to} \quad r \\ &\quad \texttt{solve} \quad U^{(\nu)} \quad \texttt{from} \quad N(U^{(\nu)} - U^{(\nu-1)}) = -N_0 U^{(\nu-1)} + C^{(k,j)} \end{aligned} \tag{14}$$

where the iteration matrix $N$ is chosen such that the linear system for the inner iterate $U^{(\nu)}$ is easily solved and where $r$ is chosen such that $U^{(r)}$ is an "acceptable" approximation to $Y_n^{(k,j)}$. Evidently, if (14) converges as $r \to \infty$, then $U^{(r)}$ converges to the solution $Y_n^{(k,j)}$ of (9) irrespective the choice for $N$. However, as we will see in the experiments, it is possible to choose "convenient" matrices $N$ such that in an actual computation, one or two inner iterations are sufficient (see Section 4). In fact, we shall define $N$ by

$$N := I \otimes K^* - T \otimes hJ^*, \tag{15}$$

where $T$ is lower triangular with positive diagonal entries (cf. [6], [8]). In order to see the intrinsic parallelism of the inner iteration process, we proceed as in the preceding section. Again assuming that $K^*$ and $J^*$ are both lower block-triangular, we obtain the (permuted) iteration matrix

$$\begin{aligned} \tilde{N} &:= (I \otimes K^*_{ij}) - (T \otimes hJ^*_{ij}) \\ &= \begin{pmatrix} I \otimes K^*_{11} - T \otimes hJ^*_{11} & O & \cdots \\ I \otimes K^*_{21} - T \otimes hJ^*_{21} & I \otimes K^*_{22} - T \otimes hJ^*_{22} & \cdots \\ \vdots & \vdots & \vdots \end{pmatrix}. \end{aligned} \tag{16}$$

Hence, (14) requires the LU decomposition of the $\sigma$ matrices $I \otimes K_{ii}^* - T \otimes hJ_{ii}^*$. But, since $T$ is also lower triangular, the LU decomposition of each of these matrices falls apart into the LU decomposition of the $s$ matrices $K_{ii}^* - T_{jj} hJ_{ii}^*$, $j = 1, \ldots, s$, which can all be done in parallel. The maximal dimension of the matrices to be decomposed equals $d^*$, so that the computational complexity is reduced to $O((d^*)^3)$, provided that $s\sigma$ processors are available (if only $p$ processors are available, with $p < s\sigma$, then effectively, the computational complexity is about $O(s\sigma p^{-1}(d^*)^3)$). Apart from these LU decompositions, each inner iteration again requires a forward/backward substitution. Furthermore, by diagonalizing $T$ by a Butcher transformation, the forward/backward substitution can be distributed over $s$ processors. If $K^*$ and $J^*$ are both *block-diagonal*, then even the forward/backward substitution can be distributed over $s\sigma$ processors.

# 3 Convergence results

In this section, we study the convergence of the inner iteration method (14) and, for linear IVPs, the convergence of the (discrete) WR iteration method (7) - (8). We recall that the outer iteration process, that is, the modified Newton process (9), is always assumed to be convergent.

## 3.1 The inner iteration method

The convergence of (14) can be studied by deriving the error recursion for $U^{(\nu)} - Y_n^{(k,j)}$, i.e.,

$$\begin{aligned} U^{(\nu)} - Y_n^{(k,j)} &= M_1 (U^{(\nu-1)} - Y_n^{(k,j)}), \\ M_1 &:= (I \otimes K^* - T \otimes hJ^*)^{-1}((A - T) \otimes hJ^*). \end{aligned} \tag{17}$$

For convergence, the spectral radius $\rho(M_1)$ of $M_1$ should be less than 1. In [8], amplification matrices of the type $M_1$ have been analysed and led to the following definition and convergence theorem:

**Definition 1** *Let*

$$Z(z) := z(I - zT)^{-1}(A - T). \tag{18}$$

*Then, $B(A)$ is the set of lower triangular matrices $T$ such that the spectrum of $Z(z)$ is within the unit circle for* $\mathrm{Re}(z) \leq 0$.

**Theorem 1** *Let $N$ be defined as in (15) with $T \in B(A)$. Then, the inner iteration process (14) converges for all $h > 0$ if, and only if, $\{K^*, J^*\}$ is stable.*

For the construction of lower triangular matrices $T$ that are in $B(A)$, we refer to [6].

## 3.2 The discrete WR iteration method

The convergence of discrete WR methods of RK type of the form (6) has extensively been studied, in particular for the ODE case where $K = K^* = I$ (see e.g., [5], [2], and the references in [2]). For linear problems, where second-order terms in the error recursion can be ignored, the convergence analysis is quite straightforward. In this section, we give a brief derivation of a few convergence results.

From (7) - (8) and (2) it follows that for linear problems the WR iteration error $Y_n^{(k)} - Y_n$ satisfies the recursion

$$\begin{aligned} &Y_n^{(k)} - Y_n = M_2(Y_n^{(k-1)} - Y_n) + M_3(Y_{n-1}^{(k)} - Y_{n-1}), \\ &M_2 := N_0^{-1}(I \otimes (K^* - K) - A \otimes h(J^* - J)), \\ &M_3 := N_0^{-1}(E \otimes K), \qquad N_0 := I \otimes K^* - A \otimes hJ^*. \end{aligned} \tag{19}$$

This recursion is of a similar form as the error recursion of the PDIRKAS GS method analysed in [7] and can be represented as

$$\varepsilon^{(k)} = Q^k \varepsilon^{(0)}, \quad \varepsilon^{(k)} := \begin{pmatrix} Y_{\kappa\omega+1}^{(k)} - Y_{\kappa\omega+1} \\ Y_{\kappa\omega+2}^{(k)} - Y_{\kappa\omega+2} \\ \vdots \\ Y_{\kappa\omega+\omega}^{(k)} - Y_{\kappa\omega+\omega} \end{pmatrix}, \tag{20}$$

$$Q := \begin{pmatrix} M_2 & O & O & O & \dots \\ M_3M_2 & M_2 & O & O & \dots \\ M_3^2M_2 & M_3M_2 & M_2 & O & \dots \\ M_3^3M_2 & M_3^2M_2 & M_3M_2 & M_2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Hence, we have convergence if the spectral radius $\rho(Q)$ of $Q$ is less than 1, i.e., if $\rho(M_2) < 1$. An estimate for $\rho(M_2)$ can be obtained along the lines of a similar approach as in [8]. Theorem 2 presents conditions for convergence using the logarithmic matrix norm $\mu[\cdot]$ associated with the Euclidean norm $\|\cdot\|$, i.e., for any square matrix $S$, we have $\mu[S] = \frac{1}{2}\lambda_{\max}(S + S^H)$, where $S^H$ is the complex transposed of $S$ and $\lambda_{max}(\cdot)$ denotes the algebraically largest eigenvalue.

**Theorem 2** *Let the IVP (1) be linear, let the spectrum $\sigma(A)$ of $A$ be in the positive halfplane, and define (if $K^*$ is nonsingular)*

$$\tilde{K} := (K^*)^{-1}K, \quad \tilde{J} := (K^*)^{-1}J, \quad \tilde{J}^* := (K^*)^{-1}J^*. \tag{21}$$

*Then, the WR iteration process (7) - (8) converges if one of the following three conditions is satisfied for all $\alpha \in \sigma(A)$:*

$$\| (K - K^*) - \alpha h(J - J^*) \| < -\mu[-K^* + \alpha h J^*], \tag{22}$$

$$\| (\tilde{K} - I) - \alpha h(\tilde{J} - \tilde{J}^*) \| < \frac{\mathrm{Re}(\alpha)}{|\alpha|} - h|\alpha|\mu[\tilde{J}^*], \ K^* \text{ nonsingular}, \tag{23}$$

$$\|(J^*)^{-1}J - I \| < \frac{\mathrm{Re}(\alpha)}{|\alpha|}, \ \mu[\tilde{J}^*] \leq 0,$$
$$K^* = K, \ K^* \text{ and } J^* \text{nonsingular}. \tag{24}$$

**Proof.** Let the eigenvectors and eigenvalues of $M_2$ be denoted by $a \otimes w$ and $\tilde{\mu}$, where $a$ is an eigenvector of $A$ with eigenvalue $\alpha$. Then,

$$(K^* - \alpha h J^*)^{-1}((K^* - K) - \alpha h(J^* - J))w = \tilde{\mu} w, \tag{25}$$

so that

$$\rho(M_2) < \| (K - K^*) - \alpha h(J - J^*) \| \, \| (K^* - \alpha h J^*)^{-1} \| . \tag{26}$$

By virtue of a property of the logarithmic norm, we have for any nonsingular, complex matrix $C$ with $\mu[-C] < 0$, the estimate $\| C^{-1} \| < -(\mu[-C])^{-1}$. Hence, if $\mu[-K^* + \alpha h J^*] < 0$, then $\| (K^* - \alpha h J^*)^{-1} \| < -(\mu[-K^* + \alpha h J^*])^{-1}$. This leads to condition (22). Note that here $K^*$ is allowed to be singular.

If $K^*$ is nonsingular, then we may write

$$\rho(M_2) < \| (\alpha^{-1} - h\tilde{J}^*)^{-1} \| \, \| \alpha^{-1}(\tilde{K} - I) - h(\tilde{J} - \tilde{J}^*) \| . \tag{27}$$

Proceeding as above, we derive (23).

Finally, we consider the case where $K^* = K$ and where both $K^*$ and $J^*$ are nonsingular. From (25) we derive the inequality

$$\rho(M_2) < \| (I - \alpha h \tilde{J}^*)^{-1}(\alpha h \tilde{J}^*) \| \, \| I \otimes ((J^*)^{-1}J - I) \| . \tag{28}$$

In this case, we use a theorem of Von Neumann. Von Neumann's theorem states that, given a matrix $X$ with $\mu[X] \leq 0$ and a rational function $R$ of $z$ which is bounded in the lefthand halfplane $\mathrm{Re}(z) \leq 0$, then with respect to the Euclidean norm, the value of $\| R(X) \|$ is bounded by the maximum of $\{|R(z)| : \mathrm{Re}(z) \leq 0\}$ (see e.g., [4], p. 179). Thus, assuming that $\mu_2[\tilde{J}^*] \leq 0$, condition (24) follows from

$$\| (I - \alpha h \tilde{J}^*)^{-1}(\alpha h \tilde{J}^*) \| \leq \max_{\mathrm{Re}(z) \leq 0} |z\alpha(1 - z\alpha)^{-1}| = \frac{|\alpha|}{\mathrm{Re}(\alpha)}. \tag{29}$$

Let us compare the convergence conditions of this theorem for the particular case where $K^* = K$. Then conditions (22), (23) and (24) simplify to

$$\| J - J^* \| < -\frac{1}{h|\alpha|}\mu[-K + \alpha h J^*], \tag{30}$$

$$\| K^{-1}(J - J^*) \| < \frac{\mathrm{Re}(\alpha)}{h|\alpha|^2} - \mu[K^{-1}J^*], \ K \text{ nonsingular}, \tag{31}$$

$$\| (J^*)^{-1}J - I \| < \frac{\mathrm{Re}(\alpha)}{|\alpha|}, \ \mu[K^{-1}J^*] \leq 0, \ K \text{ and } J^* \text{ regular}. \tag{32}$$

The conditions (30), (31) and (32) respectively provide an absolute estimate, a scaled absolute estimate and a relative estimate for the difference between $J$ and $J^*$. Note that condition (32) implies unconditional convergence with respect to $h$. For example, for the four-stage Radau IIA corrector, we have unconditional convergence if $\| (J^*)^{-1}J - I \| < 0.56$. If $A$ has its eigenvalues in the positive halfplane, then condition (31) shows that unconditional convergence is also possible if $\| K^{-1}(J - J^*) \| < -\mu[K^{-1}J^*]$.

# 4 Numerical experiments

The crucial aspect of the iteration process $\{(7), (8), (9)\}$, is the convergence behaviour for splitting functions $\psi$ for which the matrix $N_0$ allows a fast solution of the associated linear systems. Equally crucial is the effect of the number of inner and outer iterations $r$ and $m$, and the window length $\omega h$. In this section, we illustrate the performance for a few test problems.

For the predictor we chose the "last step point" formula $Y_n^{(0)} = e \otimes y_{n-1}$, and we used the four-stage Radau IIA corrector whose Butcher matrix is (within 14 digits) given by

$$A = \begin{pmatrix} .11299947932316 & -.04030922072352 & .02580237742034 & -.00990467650730 \\ .23438399574740 & .20689257393536 & -.04785712804854 & .01604742280652 \\ .21668178462325 & .40612326386737 & .18903651817006 & -.02418210489983 \\ .22046221117677 & .38819346884317 & .32884431998006 & .06250000000000 \end{pmatrix}.$$

Following [6], we choose for the matrix $T$ the lower triangular factor $L$ of the Crout decomposition $LU$ of $A$, i.e.,

$$T = L \tag{33}$$

$$= \begin{pmatrix} .11299947932312 & 0 & 0 & 0 \\ .23438399574745 & .29050212926461 & 0 & 0 \\ .21668178462320 & .48341807916606 & .30825766001501 & 0 \\ .22046221117877 & .46683683945825 & .44141588145851 & .11764705882353 \end{pmatrix}.$$

This choice implies that the amplification matrix $Z(z)$ defined in (18) becomes strictly upper triangular at infinity, i.e., $Z(\infty) = I - T^{-1}A = I - U$. As a consequence, the stiff iteration error components are strongly damped in the iteration process. Moreover, we verified numerically that the matrix $T$ given in (33) lies in $B(A)$. Hence, it follows from Theorem 1 that for each $k$ and $j$ the inner iterates $U^{(\nu)}$ in (14) unconditionally converge as $\nu \to \infty$ whenever the pair $\{K^*, J^*\}$ is stable. Note that there is no need to give the entries of $T$ with extreme accuracy. As long as $T$ lies in $B(A)$, convergence is ensured (see Definition 1).

In all experiments, constant step sizes have been used (if needed, we adapted the initial condition such that the integration starts outside the transient phase), and the matrices $K$ and $J$ were updated in each step. We recall that per update, the effective LU costs are $O((d^*)^3)$, where $d^*$ is the maximal dimension of the diagonal blocks in the matrices $K^*$ and $J^*$.
For given numbers of WR iterations $q$, outer iterations $m$, inner iterations $r$, and given window size $\omega$, the tables of results present the minimal number of correct digits cd of the components of $y$ at the end point $t = t_{\text{end}}$ of the integration interval (i.e., the absolute errors are written as $10^{-\text{cd}}$). We recall that the total number of effective inner iterations per step is given by $r_{\text{total}} = mr(1 + \omega^{-1}(q-1))$, which may serve as an estimate for the effective costs that are additional to the LU-costs. For the small window sizes used in practice and the usually large number of WR iterations needed to solve the IVP, we may approximately set $r_{\text{total}} = mrq\omega^{-1}$.

## 4.1 HIRES problem of Schäfer

Our first test problem is provided by the HIRES problem given in ([4], p. 157) which originates from Schäfer [13] for explaining the "High Irradiance Responses" of photomorphogenesis (see also Gottwald [3] and the CWI testset [11]). This problem was integrated over the interval $[5, 305]$. Writing the system as $y' = f(y)$, we may define the block-Jacobi splitting function

$$\psi(u', v', u, v) = u' - f(u) + 0.035(u_5 - v_5)e_3 + 0.69(u_4 - v_4)e_6, \tag{34}$$

with the associated Jacobian splitting $K^* = K = I$ and

$$J^* = \begin{pmatrix} J_{11} & O \\ O & J_{22} \end{pmatrix}, \qquad J - J^* = \begin{pmatrix} O & J_{12} \\ J_{21} & O \end{pmatrix}, \tag{35}$$

$$J_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.035 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad J_{21} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.69 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The results in Table 1 show that the outer iteration process converges quite fast and that even a single outer iteration already produces a relatively high accuracy. The inner iteration process converges equally fast and two inner iterations usually suffices to find the modified Newton iterate. However, the WR iteration process requires relatively many iterations to reach the corrector solution, particularly on larger windows. Furthermore, note that for a constant total number of effective inner iterations $r_{\text{total}} = mrq\omega^{-1}$, the accuracy rapidly decreases as $m$ increases (cf. Remark 2.1). Thus, the best iteration strategy seems to be one outer iteration and one or two inner iterations.

The performance of the WR iteration can be improved if we apply block-Gauss-Seidel splitting:

$$\psi(u', v', u, v) = u' - f(u) + 0.035(u_5 - v_5)e_3, \tag{36}$$
$$J^* = \begin{pmatrix} J_{11} & O \\ J_{21} & J_{22} \end{pmatrix}, \quad J - J^* = \begin{pmatrix} O & J_{12} \\ O & O \end{pmatrix}.$$

Table 2 presents the analogue of Table 1 and clearly shows the increased rate of convergence.

### 4.2 The transistor amplifier

Our second test problem is the semi-explicit representation of the transistor amplifier given in [11]. It is a nonlinear, eight-dimensional problem of index 1 on the interval $[0, 0.2]$ given by

$$u'(t) = f(u, v), \quad g(u, v) = 0, \quad u, f \in R^5, \quad v, g \in R^3, \tag{37}$$

so that

$$K = \begin{pmatrix} I & O \\ O & O \end{pmatrix}, \quad J = \begin{pmatrix} f_u & f_v \\ g_u & g_v \end{pmatrix}. \tag{38}$$

The structure of $K$ and $J$ suggests the use of a block-Gauss-Seidel splitting with

$$K^* = K, \quad J^* = \begin{pmatrix} f_u & O \\ g_u & g_v \end{pmatrix}, \tag{39}$$

which reduces the effective costs of each LU-update by a factor $8^3/5^3 \approx 4$.

The results in Table 3 show the same trends as in the preceding tables.

## 5 Summary and concluding remarks

The numerical integration method proposed in this paper is based on a Runge-Kutta type integration formula (2) which is solved iteratively by three nested iteration processes: the discrete WR process (7) - (8), the modified Newton

Table 1: WR method {(7),(8),(9),(14)} applied to HIRES with block-Jacobi spitting (34), $h = 15$ and $r = 1\backslash 2$.

| $\omega$ | $m$ | $q=3$ | $q=5$ | $q=7$ | $q=9$ | $q=11$ | $q=13$ | $q=15$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1.4 \ 1.9 | 2.6 \ 3.6 | 3.7 \ 5.7 | 4.9 \ 6.2 | 6.1 \ 7.0 | 7.8 \ 8.2 | 7.9 \ 7.9 |
| | 2 | 1.8 \ 1.9 | 3.6 \ 3.8 | 5.3 \ 6.1 | 7.1 \ 7.8 | 7.8 \ 7.9 | 7.9 \ 7.9 | |
| | 3 | 1.9 \ 1.9 | 3.8 \ 3.8 | 5.9 \ 6.1 | 7.7 \ 7.8 | 7.9 \ 7.9 | | |
| 2 | 1 | 1.0 \ 1.2 | 2.0 \ 2.6 | 3.0 \ 4.1 | 4.0 \ 6.1 | 5.1 \ 6.4 | 6.4 \ 7.9 | 7.4 \ 8.0 |
| | 2 | 1.2 \ 1.2 | 2.5 \ 2.6 | 4.0 \ 4.2 | 5.5 \ 6.0 | 7.1 \ 7.6 | 7.8 \ 7.9 | 7.9 \ 7.9 |
| | 3 | 1.2 \ 1.2 | 2.6 \ 2.6 | 4.2 \ 4.2 | 5.9 \ 6.0 | 7.5 \ 7.6 | 7.8 \ 7.9 | |
| 4 | 1 | 0.7 \ 0.8 | 1.4 \ 1.7 | 2.2 \ 2.8 | 3.0 \ 4.0 | 3.9 \ 5.6 | 4.9 \ 6.4 | 6.1 \ 6.9 |
| | 2 | 0.8 \ 0.9 | 1.7 \ 1.7 | 2.8 \ 2.8 | 4.0 \ 4.1 | 5.2 \ 5.4 | 6.6 \ 6.9 | 7.6 \ 7.8 |
| | 3 | 0.9 \ 0.9 | 1.7 \ 1.7 | 2.8 \ 2.8 | 4.1 \ 4.1 | 5.4 \ 5.4 | 6.9 \ 6.9 | 7.8 \ 7.8 |

Table 2: WR method {(7),(8),(9),(14)} applied to HIRES with block Gauss-Seidel splitting (36), $h = 15$ and $r = 1\backslash 2$.

| $\omega$ | $m$ | $q=3$ | $q=5$ | $q=7$ | $q=9$ | $q=11$ | $q=13$ | $q=15$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 3.2 \ 3.8 | 4.2 \ 4.7 | 5.1 \ 5.5 | 5.8 \ 6.3 | 6.6 \ 7.2 | 7.5 \ 8.2 | 7.9 \ 7.9 |
| | 2 | 4.2 \ 5.1 | 6.1 \ 6.6 | 8.0 \ 8.0 | 7.9 \ 7.9 | 7.9 \ 7.9 | 7.9 \ 7.9 | |
| | 3 | 5.1 \ 5.9 | 7.9 \ 8.0 | 7.9 \ 7.9 | | | | |
| 2 | 1 | 3.1 \ 3.6 | 4.1 \ 4.6 | 4.9 \ 5.4 | 5.6 \ 6.2 | 6.4 \ 7.0 | 7.2 \ 8.1 | 7.9 \ 7.9 |
| | 2 | 4.1 \ 5.2 | 5.8 \ 6.3 | 7.4 \ 8.2 | 7.9 \ 7.9 | 7.9 \ 7.9 | 7.9 \ 7.9 | |
| | 3 | 5.3 \ 4.7 | 7.1 \ 8.2 | 7.9 \ 7.9 | | | | |
| 4 | 1 | 3.1 \ 3.5 | 3.7 \ 4.3 | 4.6 \ 5.1 | 5.2 \ 5.8 | 5.9 \ 6.6 | 6.6 \ 7.4 | 7.4 \ 7.9 |
| | 2 | 4.2 \ 4.2 | 5.2 \ 5.7 | 6.7 \ 7.2 | 7.9 \ 7.9 | 7.9 \ 7.9 | 7.9 \ 7.9 | 7.9 \ 7.9 |
| | 3 | 4.1 \ 4.0 | 6.0 \ 6.5 | 7.9 \ 7.9 | | | | |

process or outer iteration process (9), and the linear system solver or inner iteration process (14). It aims at the solution of IDEs of which the Jacobian matrices $K$ and $J$ are approximated by lower triangular $\sigma \times \sigma$ block matrices $K^*$ and $J^*$. On $\omega\sigma s$ processors, the total effective costs per step approximately consists of carrying out $r_{\text{total}} = mrq\omega^{-1}$ inner iterations. Here, $\omega$ is the window length and $q$, $m$ and $r$ respectively denote the number of WR iterations, outer iterations and inner iterations. Each Jacobian update or change of step size requires $s$ concurrent LU-decompositions of matrices of maximal dimension $d^*$, where $d^*$ is the maximal blocksize occurring in $K^*$ and $J^*$, that is, effectively only $O((d^*)^3)$ operations per update. Furthermore, each inner iteration requires a forward/backward substitution of dimension $\leq sd^*$ which can be distributed over $s$ processors, that is, only $O(r_{\text{total}}(d^*)^2)$ operations per step.

Table 3: WR method {(7),(8),(9),(14)} applied to the Transistor amplifier with the splitting (39), $h = 2 \cdot 10^{-4}$ and $r = 1\backslash 2$.

| $\omega$ | $m$ | $q=3$ | $q=5$ | $q=7$ | $q=9$ | $q=11$ | $q=13$ | $q=15$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0.9 \ * | 1.0 \ 1.7 | 1.8 \ 3.3 | 2.8 \ 4.8 | 3.8 \ 5.9 | 4.9 \ 7.3 | 6.0 \ 8.7 |
| | 2 | 0.8 \ 1.4 | 2.7 \ 3.0 | 4.7 \ 5.1 | 5.7 \ 6.8 | 7.5 \ 8.3 | 9.3 \ 9.5 | 9.6 \ 9.8 |
| | 3 | 1.4 \ 1.2 | 2.7 \ 2.7 | 4.7 \ 4.4 | 6.8 \ 6.8 | 8.2 \ 8.1 | 9.6 \ 9.6 | 9.7 \ 9.7 |
| 2 | 1 | 0.3 \ 0.4 | 0.2 \ 0.3 | 0.3 \ 0.6 | 0.5 \ 1.0 | 0.7 \ 1.7 | 0.9 \ 2.3 | 1.2 \ 2.1 |
| | 2 | 0.6 \ 0.8 | 2.7 \ 1.8 | 2.6 \ 2.8 | 2.7 \ 3.8 | 3.2 \ 4.9 | 3.8 \ 5.8 | 4.3 \ 6.5 |
| | 3 | 0.3 \ 0.3 | 0.9 \ 1.5 | 2.0 \ 2.0 | 2.2 \ 3.1 | 3.1 \ 4.3 | 5.1 \ 5.1 | 4.5 \ 7.6 |
| 4 | 1 | * \ 0.2 | 0.7 \ 0.4 | 0.4 \ 0.6 | 0.5 \ 0.9 | 0.6 \ 1.2 | 0.7 \ 1.6 | 0.9 \ 2.0 |
| | 2 | 0.5 \ 0.7 | * \ 0.7 | * \ 1.7 | * \ 1.8 | * \ 2.1 | * \ 3.0 | * \ 4.1 |
| | 3 | 0.6 \ 0.4 | 0.7 \ 1.0 | 2.0 \ 1.8 | 2.0 \ 2.2 | 2.4 \ 3.1 | 3.1 \ 4.3 | 4.9 \ 4.7 |

The numerical experiments with the method $\{(7),(8),(9),(14)\}$ presented in Section 4 clearly show:

- The better the approximations $K^*$ and $J^*$, the faster the convergence of the WR iterates.
- One or two inner iterations are sufficient, i.e., $r \leq 2$.
- For constant $r_{\text{total}}$, the accuracy is best if only one outer iteration is performed, i.e., $m = 1$.

In an actual implementation, the values of $q$, $m$ and $r$ should be determined dynamically during the integration process. At present, the full method $\{(7),(8),(9),(14)\}$ is tested on a sequential computer system and only the case where $\omega = r = 1$ and $K = K^* = I$, $J^* = J$ (and hence $q = 1$) has been implemented on the four-processor Cray-C98 / 4256. The results reported in [6] show that with respect to the code RADAU5 of Hairer and Wanner [4], to be considered as one of the best sequential codes, the speed-ups are in the range [2.4, 3.1]. Implementation of the full method $\{(7),(8),(9),(14)\}$ on the Cray-C98 / 4256 will be subject of future research.

# References

[1] A. Bellen and F. Tagliaferro. A combined WR-parallel steps method for ordinary differential equations. In *Parallel Computing: Achievements, Problems and Prospects*, Capri, Italy, 1990.

[2] K. Burrage. *Parallel and sequential methods for ordinary differential equations.* Clarendon Press, Oxford, 1995.

[3] B. A. Gottwald. MISS - ein einfaches simulations-system für biologische und chemische prozesse. *EDV in Medizin und Biologie*, 3:85–90, 1977.

[4] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Springer-Verlag, 1996.

[5] K.J. in ' t Hout. On the convergence of waveform relaxation methods for stiff nonlinear ordinary differential equations. *Appl. Numer. Math.*, 18:175–190, 1995.

[6] P. J. van der Houwen and J. J. B. de Swart. Triangularly implicit iteration methods for ODE-IVP solvers. Technical Report NM-R9510, CWI, Amsterdam, 1995. To appear in: SIAM Journal on Scientific Computing, 18(1), January 1997.

[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] P. J. van der Houwen and W. A. van der Veen. Solving implicit differential equations on parallel computers. Technical Report NM-R9526, CWI, Amsterdam, 1995. Submitted for publication.

[9] E. Lelarasmee. *The waveform relaxation method for the time domain analysis of large scale nonlinear dynamical systems.* PhD thesis, Univ. of California, Berkeley, 1982.

[10] E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli. The waveform relaxation method for time domain analysis of large scale integrated circuits. *IEEE Trans. CAD IC Syst.*, 1:131–145, 1982.

[11] W. M. Lioen, J. J. B. de Swart, and W. A. van der Veen. Test set for IVP solvers. Report NM-R9615, CWI, Amsterdam, 1996. WWW version available at URL http://www.cwi.nl/cwi/projects/IVPtestset.shtml.

[12] U. Miekkela and O. Nevanlinna. Convergence of dynamic iteration methods for initial value problems. , *SIAM J. Sci. Statist. Comput.*, 8:459–482, 1987.

[13] E. Schäfer. A new approach to explain the 'high irradiance responses' of photomorphogenesis on the basis of phytochrome. *J. of Math. Biology*, 2:41–56, 1975.

[14] J.K. White and A.L. Sangiovanni-Vincentelli. *Relaxation techniques for the simulation of VLSI circuits.* Kluwer, Dordrecht, 1987.

# Samenvatting

Dit proefschrift handelt over methoden voor het numeriek oplossen van gewone en impliciete differentiaalvergelijkingen. We zullen vooral stijve differentiaalvergelijkingen beschouwen, waarvoor impliciete methoden vereist zijn. Uitgaande van al bestaande methoden worden nieuwe methoden ontwikkeld die speciaal bedoeld zijn voor gebruik op parallelle computers.

Voor het numeriek integreren van stijve differentiaalvergelijkingen zijn de impliciete Runge-Kutta-methoden (IRKs) en de Backward Differentiation Formulas (BDFs) het bekendst. Om in een aantal punten op de tijdas een benadering voor de oplossing te verkrijgen, moet in elk van deze punten een niet-lineair systeem worden opgelost. Voor IRK-methoden is de dimensie van dit stelsel een veelvoud van de probleemdimensie, daar deze methoden ook in een aantal tussenpunten benaderingen voor de oplossing bepalen. Voor BDF methoden is de dimensie van het niet-lineaire stelsel gelijk aan de probleemdimensie.

De standaardprocedure voor het oplossen van niet-lineaire systemen is het Newton-iteratieproces, waarbij een reeks lineaire systemen moet worden opgelost. Vaak zijn de kosten hiervan dominant en zijn voor de gangbare IRK-methoden aanzienlijk groter dan voor de BDF-methoden. Dit is ook de reden dat in de industrie IRKs zelden en BDFs vaak gebruikt worden, ondanks de beduidend betere stabiliteitseigenschappen van de IRKs. Dit alles geldt voor sequentiële computers, maar niet voor parallelle computers, daar de BDF methoden geen intrinsiek parallellisme toestaan terwijl IRKs dat wel doen. Het is mogelijk gebleken om een methode te ontwikkelen die alle goede eigenschappen van IRKs bezit en op een parallelle computer zelfs sneller is dan de BDFs.

De lineaire systemen die bij de toepassing van IRKs in elk van deze Newton-iteraties opgelost moeten worden zijn zo duur omdat ze niet ontkoppeld kunnen worden. Echter, door het idee van relaxatie toe te passen, kan het lineaire systeem met een apart iteratieproces worden opgelost waar alleen nog maar ontkoppelde lineaire systemen in voorkomen. Wanneer in dit aparte iteratieproces, maar één iteratie wordt uitgevoerd, dan verkrijgt men de zogenaamde PDIRK-methode (Parallel Diagonal implicitly-Iterated RK-method) die in 1991 geïntroduceerd werd door Van der Houwen en Sommeijer. Door deze ontkoppeling kunnen de lineaire systemen parallel over de tussenpunten worden opgelost. Voor de PDIRK-methode werd ten opzichte van de beste sequentiële code voor de Alliant FX/4 een speedupfactor van ongeveer twee gevonden.

De PDIRK-methode is al efficiënt, maar het bleek dat het aantal iteraties per stap tamelijk hoog is (ongeveer gelijk aan de orde). Om hier wat aan te doen, kan men of het idee van stap-parallellisme toepassen, of men kan de PDIRK-methode gaan verfijnen. In de hoofdstukken 2 tot en met 5 bestuderen

we stap-parallellisme voor PDIRK. In de hoofdstukken 6 en 7 beschrijven we verfijningen van de PDIRK-methode voor multistap RK-methoden en voor impliciete differentiaalvergelijkingen (zie ook het proefschrift van J.J.B. de Swart getiteld "Parallel Software for Implicit Differential Equations"). Tenslotte zullen we in Hoofdstuk 8 waveformrelaxatie beschouwen.

Het idee achter stap-parallellisme is toepasbaar voor de meeste methoden voor het numeriek oplossen van differentiaalvergelijkingen. Stel dat men in een aantal punten op de tijdas een benadering voor de oplossing wil bepalen. De meeste methoden leiden dan tot een reeks niet-lineaire systemen, voor ieder tijdspunt één. Ook zal de benadering in een tijdspunt afhangen van de benadering in het vorige tijdspunt. Deze systemen worden meestal iteratief opgelost, bijvoorbeeld met fixed-point-iteratie in het geval van niet-stijve problemen of met PDIRK in het geval van stijve problemen. Gewoonlijk worden deze systemen na elkaar opgelost: men start het iteratieproces in een bepaald punt pas als het iteratieproces in het vorige tijdspunt is afgesloten. Bij stap-parallellisme echter begint men al met het oplossen van het systeem in een bepaald tijdspunt als in het vorige tijdspunt het oplossen van het systeem nog in volle gang is. Zo kan men in een tijdspunt de iteratie starten zodra de tussentijdse benadering in het vorige tijdspunt voldoende betrouwbaar is. Op deze wijze kunnen iteraties in verscheidene tijdspunten tegelijkertijd worden uitgevoerd. In de hoofdstukken 2 en 3 wordt dit idee uitgewerkt voor respectievelijk fixed-point-iteratie en de PDIRK-methode.

De in hoofdstuk 2 beschreven experimenten laten speedupfactoren zien van rond de vijf.

In hoofdstuk 3 passen we stap-parallellisme toe op de PDIRK methode, waarbij we ons beperken tot het gebruik van vaste staplengten. Enige experimenten met eenvoudige testproblemen laten een speedupfactor zien van circa drie.

De vraag doet zich voor of PDIRKAS (PDIRK with parallelism Across the Steps) wel altijd convergeert. In hoofdstuk 4 tonen we aan dat dit het geval is. Hierbij maken we gebruik van Fourieranalyse, complexe analyse en van het begrip $\varepsilon$-pseudo-spectrum.

Om een robuuste en efficiënte code voor PDIRKAS te verkrijgen is het belangrijk dat het aantal punten dat tegelijkertijd word behandeld goed wordt bepaald. In hoofdstuk 5 beschrijven we hiervoor een algoritme, die bepaalt wanneer welke punten worden behandeld en die tevens met variabele staplengten werkt. Experimenten met realistische testproblemen laten zien dat de algoritme voldoende robuust is en een speedup van tenminste twee oplevert ten opzichte van PDIRK.

Hoofdstuk 6 betreft multistap RK-methoden. Deze methoden lijken veel op IRK-methoden en ook hier loont het de lineaire systemen in de Newton-iteraties

iteratief op te lossen.

In hoofdstuk 7 en 8 beschouwen we impliciete differentiaalvergelijkingen. Dit kunnen bijvoorbeeld differentiaal-algebraïsche vergelijkingen zijn. Dit zijn stelsels vergelijkingen die deels uit differentiaalvergelijkingen en deels uit algebraïsche vergelijkingen bestaan. Bij het oplossen van impliciete differentiaalvergelijkingen krijgt men ook weer te maken met niet-lineaire systemen die eveneens met een iteratief proces kunnen worden opgelost. We beschrijven in hoofdstuk 7 enkele iteratieschema's, die betere convergentie eigenschappen hebben dan PDIRK.

Het laatste hoofdstuk handelt over waveformrelaxatie, dat veel toepassing vindt binnen de circuitanalyse. Het idee is dat men in plaats van het originele, complexe probleem een reeks van betrekkelijk eenvoudige problemen gaat oplossen. Deze reeks is zo ontworpen dat als de reeks van oplossingen convergeert, het convergeert naar de oplossing van het oorspronkelijke probleem. Bij het gebruik van IRK-methoden binnen deze waveformrelaxatiemethoden krijgt men weer te maken met niet-lineaire stelsels waar de PDIRK-gedachte weer op van toepassing is. Helaas blijkt hier de PDIRK-methode niet te werken, maar een verfijning die beschreven wordt in het proefschrift van J.J.B. de Swart blijkt goed te werken.