Parallel Software for Implicit Differential Equations

Parallel Software for 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 vrijdag 28 november 1997 te 13.30 uur

door

Jacques J.B. de Swart

geboren in De Kwakel in 1969

Parallelle Software voor Impliciete Differentiaalvergelijkingen

Promotor:	Prof. Dr P.J. van der Houwen
Co-promotor:	Dr W. Hoffmann
Faculteit:	Wiskunde, Informatica, Natuurkunde en Sterrenkunde

Het hier beschreven onderzoek is gefinancierd door de Technologiestichting STW onder projectnummer CWI22.2703.

Preface

Systems of implicit differential equations often arise when time-dependent processes are modeled. Since these equations are normally too difficult to be solved exactly, one often uses a computer code to find an approximation to the solution. Although for conventional computers there are many good codes of this type available, this is not the case for computers with more than one processor. Especially there is a lack of algorithms for which the way of dividing the work over the processors is independent of the differential equations to be solved.

To fill this gap, the Centre for Mathematics and Computer Science in Amsterdam initiated in 1993 the project 'Parallel Codes for circuit analysis and control engineering', which was financed by the Dutch Technology Foundation STW over a period of four years. The underlying thesis is one of the results of this project. The other main deliverables are the PhD thesis by W.A. van der Veen [Vee97], a number of real-life test problems collected in [LSV96] and the code PSIDE (abbreviating the title of this thesis).

Chapter 1 serves as an introduction for the non-expert reader and describes the scope of the subsequent chapters along rough lines. Chapter 2–9 are copies of published journal papers, which are slightly adapted to achieve a uniform representation and to enable cross-referring within the thesis. Chapter 10 is based on a technical report that has been submitted for publication. It includes numerical results obtained with PSIDE, that were not available at the time the report was printed.

The papers are mostly co-productions and listed below.

Chapter 2 P.J. van der Houwen, B.P. Sommeijer, and J.J.B. de Swart. Parallel predictor-corrector methods. *Journal of Computational and Applied Mathematics*, 66:53–71, 1996.

Chapter 3 J.J.B. de Swart. Efficient parallel predictor-corrector methods. *Applied Numerical Mathematics*, 18:387–396, 1995.

Chapter 4 P.J. van der Houwen and J.J.B. de Swart. Triangularly implicit iteration methods for ODE-IVP solvers. *SIAM Journal on Scientific and Statistical Computing*, 18(1):41–55, 1997.

Chapter 5 W. Hoffmann and J.J.B. de Swart. Approximating Runge–Kutta matrices by triangular matrices. *BIT Numerical Mathematics*, 37(2):346–354, 1997.

Chapter 6 J.J.B. de Swart and J.G. Blom. Experiences with sparse matrix solvers in parallel ODE software. *Computers & mathematics with applications*, 31(9):43–55, 1996.

Chapter 7 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.

Chapter 8 E. MESSINA, J.J.B. DE SWART, AND W.A. VAN DER VEEN. Parallel iterative linear solvers for multistep Runge–Kutta methods. *Journal of Computational and Applied Mathematics*, 85(1):145–167, 1997.

Chapter 9 J.J.B. DE SWART AND G. SÖDERLIND. On the construction of error estimators for implicit Runge–Kutta methods. To appear in *Journal of Computational and Applied Mathematics.*

Chapter 10 J.D. PINTÉR, W.J.H. STORTELDER, AND J.J.B. DE SWART. Computation of elliptic Fekete point sets. Report MAS-R9705, CWI, Amsterdam, 1997. *Submitted for publication.*

Chapter 11 and 12 contain the specification and users' guide of PSIDE. W.M. Lioen and W.A. van der Veen are co-authors of these chapters.

A collection of mutually related papers implies that there is some overlap in the first sections of each chapter. However, the advantage is that the reader who is interested in a particular chapter can read this chapter separately, and only has to go through the introductionary material that directly applies to the subject.

Contents

Chapter 1

Introduction and outline

1.1 General scope

For many processes in industry the solution of differential equations is indispensable. For example, to test the design of a computer chip, its behavior is modeled by a set of differential equations. If the solution of these equations satisfies the requirements, then the chip is manufactured; if not, then the design is to be adjusted. Thus the production process becomes much cheaper than in the case where all designs—including the wrong ones—are first manufactured and then tested. Other examples that can be modeled by differential equations are the behavior of a train on a rail track, the steering of robots and chemical reactions.

Normally the differential equations are far too complicated to solve analytically, and one has to resort to numerical integration techniques implemented on a computer to obtain an approximation to the solution. Both the increase of the complexity of the problems—chips are getting more and more complicated—as well the demand for a decrease in computer run time—it is not desirable to steer the motion of a robot by software that needs several seconds before a new move has been computed—have challenged the numerical analysts to come up with faster algorithms. Much can be gained by making the algorithms suitable for implementation on modern computer architectures that contain more than one processor, so-called *parallel* computers. To increase the speed of the fastest state-of-the-art processor becomes more difficult and costly, whereas the price of simpler, but still reasonably fast processors has dropped considerably over the years. This development inspired many computer companies to start the production of parallel computers.

The differential equations arising from the modeling process may have different forms. The most simple formulation of interest here is that of the Initial Value Problem (IVP) for Ordinary Differential Equations (ODEs), which reads: Given a function $f : \mathbb{R}^d \to \mathbb{R}^d$, find the function $y : \mathbb{R} \to \mathbb{R}^d$ that fulfills

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad t_0 \le t \le t_{\text{end}}. \tag{1.1}$$

Almost every method to solve (1.1) numerically is a step-by-step method; one divides the interval $[t_0, t_{\text{end}}]$ in subintervals $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{N-1}, t_N]$, where $t_N = t_{\text{end}}$, and computes approximations y_1, y_2, ..., y_N to the solution at the end of each subinterval. The accuracy of the method will depend on the length of the subintervals, which we call the *stepsize* and denote by h. If $y_N - y(t_{\text{end}})$ is $\mathcal{O}(h^p)$, then the *order* of the method is p.

The computation of y_n in a conventional step-by-step method depends on approximations in time points prior to t_n; to proceed in time information from the past has to be available. This means that the numerical solution process is to a large extent sequential by its nature and offers little scope for parallelization. Nevertheless, several attempts have been made to exploit parallel computer architectures, which has been categorized by Gear in three classes: *(i)* parallelism across the problem, *(ii)* parallelism across the time and *(iii)* parallelism across the method. The first class consists of rather obvious ways to distribute the various components of the system of ODEs amongst the available processors. Since the project of which this thesis is one of the deliverables, is aiming at multi-purpose solvers, a problem dependent approach as *(i)* will be excluded. For methods based on parallelism across the time we refer to the other thesis written as a result of this project, [Vee97]. The present thesis deals with solution techniques belonging to the third category, which means that we want to employ parallelism inherently available within a method. For example, the method may be such that the computation of y_n requires several evaluations of f that can be done concurrently. Notice that this form of parallelism may even be effective for scalar problems (i.e. $d = 1$ in (1.1)), whereas approach *(i)* requires high d-values.

In particular we investigate Runge–Kutta (RK) methods which are adapted such that the so-called *stages* are computed in parallel. First we briefly resume some terminology of RK methods.

Runge–Kutta methods

We write a Runge–Kutta method in the following form:

$$Y_n = \mathbb{1} \otimes y_{n-1} + h(A \otimes I)F(Y_n)\,, \tag{1.2}$$

$$y_n = y_{n-1} + h(b^{\mathrm{T}} \otimes I)F(Y_n)\,. \tag{1.3}$$

Here, Y_n is the so-called *stage vector*, which contains s approximations $Y_{n,i}$, $i = 1, 2, \ldots, s$, to the solution in the time points $t_{n-1} + c_i h$, i.e., $Y_n = (Y_{n,1}^{\mathrm{T}}, Y_{n,2}^{\mathrm{T}}, \ldots, Y_{n,s}^{\mathrm{T}})^{\mathrm{T}}$, where $Y_{n,i} \approx y(t_{n-1} + c_i h)$. The scalars c_i determine where the solution is approximated and are called the *abscissae*. The length of the subinterval $[t_{n-1}, t_n]$ is the *stepsize* and is denoted by h. The symbol $\otimes$ denotes the direct product, which is defined by

$$\begin{bmatrix} v_{11} & \cdots & v_{1l} \\ \vdots & & \vdots \\ v_{k1} & \cdots & v_{kl} \end{bmatrix} \otimes W = \begin{bmatrix} v_{11}W & \cdots & v_{1l}W \\ \vdots & & \vdots \\ v_{k1}W & \cdots & v_{kl}W \end{bmatrix},$$

where $V = (v_{ij})$ and W are matrices of arbitrary dimensions. Furthermore, $\mathbb{1}$ stands for the s-dimensional vector $(1, \ldots, 1)^{\mathrm{T}}$, the identity matrix of the problem dimension d is

denoted by I and $F(Y_n)$ means the componentwise f-evaluation, i.e.

$$F(Y_n) = \begin{pmatrix} f(Y_{n,1}) \\ \vdots \\ f(Y_{n,s}) \end{pmatrix},$$

so that $F(Y_n)$ is of dimension sd. The $s \times s$ matrix A and the s-dimensional vector b contain the parameters of the Runge–Kutta method. If the matrix A is full, then we call (1.2)–(1.3) an Implicit Runge–Kutta method (IRK). In most cases the function f in (1.1) is non-linear, which implies that for an IRK the sd-dimensional system (1.2) is non-linear. Once Y_n is solved from this system, we can compute the approximation to the time point t_n by formula (1.3).

To select the type of RK method and the strategy to solve Y_n from the non-linear system, the notion of stiffness is important. If the time scales of the various solution components greatly vary, then we call a problem *stiff*. For example, if both high and low frequency signals play a role in an electrical circuit, then the modeling of such a circuit gives rise to a stiff system of differential equations. Such a problem imposes severe stability demands on the numerical method. We start with the treatment of non-stiff problems.

1.2 Non-stiff problems

An interesting generalization of an RK method is the family of methods of the from

$$\begin{aligned} Y_n &= (A \otimes I)Y_{n-1} + h(B \otimes I)F(Y_{n-1}) + h(C \otimes I)F(Y_n)\,, \\ y_n &= y_{n-1} + h(b^{\mathrm{T}} \otimes I)F(Y_n)\,, \end{aligned} \tag{1.4}$$

where A, B and C are matrices of dimension $s \times s$ and contain method parameters. This class of methods is similar to the RK method (1.2)–(1.3), but in the formula for Y_n we now use information of the previous stage vector Y_{n-1} and its f-evaluation as well. This raises the order of the method, at the price of reduced stability, which is not harmful for non-stiff problems. We may use so-called *fixed-point iteration* to find Y_n. Given some initial guess $Y_n^{(0)}$, we define a sequence of iterates by

$$Y_n^{(j)} = (A \otimes I)Y_{n-1} + h(B \otimes I)F(Y_{n-1}) + h(C \otimes I)F(Y_n^{(j-1)})\,, \tag{1.5}$$

and accept $Y_n^{(m)}$ as approximation for Y_n if it fulfills (1.4) accurately enough. The method for determining $Y_n^{(0)}$ is called the *predictor* and (1.5) the *corrector*. In Chapter 2 and 3 we will develop predictor–corrector methods, in which several stages can be computed in parallel.

1.3 Stiff problems

Since many applications treated in the aforementioned project yield stiff systems of differential equations, the other chapters of these thesis deal with numerical solution

techniques capable of handling stiffness. Formula (1.4) is not stable enough in this case and we stick to Formula (1.2), in which the matrix A is full. Furthermore, fixed-point iteration may cause severe stepsize restrictions for stiff problems and can not be used anymore. A well-known alternative is the modified Newton process, which takes the form

$$(I - A \otimes hJ)(Y_n^{(j)} - Y_n^{(j-1)}) = -R(Y_n^{(j-1)}), \tag{1.6}$$

where, for any $X \in \mathbb{R}^{sd}$, $R(X) = X - \mathbb{1} \otimes y_{n-1} - h(A \otimes I)F(X)$, and J stands for an approximation to the Jacobian of f evaluated in $y(t_{n-1})$, i.e.,

$$J \approx \frac{\partial f}{\partial y}(y(t_{n-1})).$$

Again $Y_n^{(0)}$ is produced by a predictor formula and (1.6) is applied as many times as needed to make $Y_n^{(j)}$ sufficiently close to the true solution of (1.2).

The dimension of the linear system (1.6) is sd, which makes IRKs relatively expensive to implement. For this reason they are not frequently used in practice. Most industrial codes for stiff problems are based an Backward Differentiation Formulas (BDFs), which require every Newton iteration the solution of linear systems only of dimension d. On the other hand, BDFs do not allow for parallelism across the method and from the famous Dahlquist order barrier it follows that having high order and being unconditionally stable are two properties that can not be combined by BDFs. Another disadvantage is that the BDF of order k is a k-step method; it bases the approximation y_n on information collected in the previous k subintervals. Evidently, this complicates the change of stepsizes. Moreover, if for some reason the method has to be restarted frequently, e.g. due to discontinuities in the function f, then in every restart one has to apply the one-step BDF of first order and 'to build up' the order in the subsequent steps.

For IRKs the situation is opposite. From (1.2)–(1.3) it is clear that these are one-step methods and, although expensive to implement on a sequential computer (a computer with only one processor), IRKs can benefit from parallelism across the method and are unconditionally stable. The task we are now faced with is to make IRK methods suitable for implementation on parallel computers. We will present two approaches for parallelism across the IRK method. The first is based on the solution of *non-linear* systems, the second on the solution of *linear* systems.

1.4 Parallel non-linear system solvers

The first approach is based on a direct treatment of the non-linear equation (1.2). In the modified Newton process, J is not more than just an *approximation* to the true Jacobian $\frac{\partial f}{\partial y}(y(t_{n-1}))$. In practice, a code often works most efficiently if J is even 'frozen' for a number of time steps. This means that a Newton process with a 'not so accurate' matrix $I - A \otimes hJ$ may still perform well. Based on this observation one can think of 'approximating' the matrix A as well, where the approximated A is such that it reduces

the computational costs by the use of more processors. Along these lines we propose in Chapter 4 to solve (1.2) by the Newton-like iteration process

$$(I - T \otimes hJ)(Y_n^{(j)} - Y_n^{(j-1)}) = -R(Y_n^{(j-1)}), \tag{1.7}$$

where T is a lower triangular matrix, which approximates A in some sense. It will turn out that (1.7) can be diagonalized such that a stage in $Y_n^{(j)}$ can be solved from a linear system of dimension d, independently from the other stages, thereby making the process suitable for implementation on s processors.

In Chapter 5 we prove that the procedure for selecting T is such that the diagonal entries of T are positive, which is for most problems of interest a necessary and sufficient condition for the matrix $I - T \otimes hJ$ to be regular.

1.5 Parallel linear solvers

In our second approach we solve the *sd*-dimensional *linear* system (1.6) by a second iteration process, i.e., we compute a sequence of iterates

$$Y_n^{(j,0)}, Y_n^{(j,1)}, \ldots,$$

which converges to $Y_n^{(j)}$. In Chapter 7 this iteration process is constructed such that the s components of dimension d in $Y_n^{(j,\nu)}$ can be solved concurrently from linear systems of dimension d. We achieve this objective by first writing the matrix A as $B+(A-B)$, then bringing the term containing $A - B$ to the right-hand side of the equation, and finally applying fixed-point iteration, thus yielding

$$(I-B \otimes hJ)(Y_n^{(j,\nu)} - Y_n^{(j-1)}) = ((A-B) \otimes hJ)\,(Y_n^{(j,\nu-1)} - Y_n^{(j-1)}) - R(Y_n^{(j-1)}), \tag{1.8}$$

This process can be seen as a splitting method as well. We select B in (1.8) such that there exists a transformation matrix Q with the property that $Q^{-1}AQ$ is a block diagonal matrix and $Q^{-1}BQ$ is a block diagonal matrix with diagonal blocks that are lower triangular. It will be shown that it is possible to transform the whole process (1.8) to a process that allows the desired decoupling of stages.

Although this approach is more complicated than that presented in Chapter 4, the advantage is twofold; the resulting method is more efficient for many applications and we can derive strong theoretical results on the convergence of the iteration process.

Chapter 8 discusses several variants of this construction applied to a somewhat broader class of methods, that of the Multistep Runge–Kutta methods.

1.6 Linear algebra

The parallel solvers for non-linear and linear systems of dimension *sd* presented in this thesis both evolve in linear systems of dimension d, and the performance of the overall method strongly depends on how efficient these systems are solved. Although for

reasons of simplicity we will often assume that Gaussian elimination is used as linear solver, it is possible to apply any linear system solver, such as QR-decomposition or a Krylov subspace method. In this work we do not focus on this aspect, except in Chapter 6, which treats the subject of linear solvers for systems with sparse matrices. For problems arising in practice it often occurs that the components of the function f only depend on a small number of variables. Consequently, for these problems the Jacobian J contains many zeros; we call J a *sparse* matrix. After reviewing a few off-the-shelf linear solvers that exploit sparsity, we come up with a new sparse matrix solver, that works very efficiently because it is tuned to the structure of the linear systems arising from numerical procedures for solving stiff differential equations.

1.7 Software development

The goal of the underlying project was not only to construct parallel numerical methods, but also to develop a piece of software that incorporates these methods. It is a long road from method to software. First of all, one has to decide for which problem class the code should be written. Many applications in the project require a much broader formulation than (1.1). For example, to model the behavior of a train on a rail track, one has to add algebraic (meaning not containing differentials) side conditions, which state that the train and rail track can not intersect. The resulting system is called a set of Differential–Algebraic Equations (DAEs). We chose to make our software code suitable for the even broader class of Implicit Differential Equations (IDEs), which are of the form

$$\begin{array}{lll} g(t,y,y') = 0\,, & g : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d\,, & y : \mathbb{R} \to \mathbb{R}^d\,, \\ t_0 \le t \le t_{\text{end}}\,, & y(t_0) = y_0\,, & y'(t_0) = y'_0\,, \end{array} \tag{1.9}$$

where some components of g may contain differentials and some not. This choice explains the title of the code, which coincides with the title of this thesis: PSIDE, Parallel Software for Implicit Differential Equations.

A complication of IDEs, which does not apply to ODEs, is that some components of the IDE solution may be more sensitive to perturbations. These components are said to be of *higher index*. Although PSIDE can not solve all higher-index IDEs, it works well for certain classes of higher-index problems that cover many applications of interest.

The underlying method of PSIDE is the Radau IIA method with four stages, because this IRK combines high accuracy with excellent stability properties. We use the modified Newton process to solve the non-linear systems, which result in linear systems which are solved with the parallel linear system solver presented in Chapter 7. The number of four processors thus means that the code is suitable for implementation on four processors, which is a common number for many computer architectures.

Many other questions have to be answered when implementing these techniques. E.g., how to form a prediction for the Newton process, when to evaluate the Jacobian, how many Newton iterations should be done, is the error conducted in one time step small enough, how to vary the stepsize? We took most answers to these questions from the

literature and describe them in Chapter 11, which gives a detailed description of PSIDE. For the estimation of the error, the literature did not provide a satisfactory method and in Chapter 9 we design our own error estimator.

Chapter 12 is the user's guide of PSIDE, which gives a complete description of the user interface. It also lists the classes of higher-index problems that can be handled by PSIDE, and how the user should supply the index related information to the code.

Of course, one would like to see the gain in practice of all the methods presented in this thesis; in other words, one wants to get insight in the performance of PSIDE compared to existing solvers when applied to a wide range of real-life problems. Proper testing of software is a whole field by itself and is not discussed in this thesis. We refer to another deliverable of the project, namely the Test set for IVP solvers [LSV96], in which a large number of problems from both industry and literature are brought together and solved by off-the-shelf solvers and PSIDE. However, we give some test results in Chapter 12, when we discuss the relation between PSIDE and this test set.

Finally, Chapter 10 describes one application in detail, the computation of elliptic Fekete point sets. Although for the modeling process of many real-life problems we again refer to [LSV96], we think it is important to explain the modeling of this problem in this thesis because originally the elliptic Fekete point sets was a challenging problem in the field of global optimization. Surprisingly, it can be shown that modeling by a system of implicit differential equations is possible as well. Since the system is rather complicated and of high dimension, it is an outstanding example of an IDE to be solved by PSIDE. We compare the results of PSIDE with the outcome of a global optimization package, and it will turn out that, although PSIDE uses more memory, it is favorable in terms of accuracy and computer time.

Chapter 2

Parallel predictor–corrector methods

Abstract In this chapter, we construct predictor–corrector methods using block Runge–Kutta methods as correctors. Like conventional Runge–Kutta methods, these correctors compute stage values at non-uniformly distributed, intermediate points. The predictor–corrector nature of the methods makes these suitable for implementation on parallel computers. Comparisons of an 8^{th}-order, 5-processor predictor–corrector method using Radau II points with the celebrated 8(7) Runge–Kutta method of Prince and Dormand show speed-up factors from 1.9 until 2.9.

2.1 Introduction

We shall consider predictor–corrector methods (PC methods) for solving the (non-stiff) initial value problem (IVP)

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \leq t \leq t_{\text{end}} \tag{2.1}$$

on parallel computers. On one-processor computers, PC methods based on linear multistep (LM) methods of Adams-type are most widely used. However, the use of multiprocessor computers enables us to apply PC methods with a much more powerful corrector than in the conventional one-processor PC methods. The general characteristic of these correctors is that they relate whole *blocks* of solution values with each other, rather than single solution values (as in classical LM methods). This has already been observed and tried out in a number of papers. For example, in [BAR87, CH87] correctors have been constructed where the solution values in each block are equidistant (like LM methods), and in [HS90, HS92, JN95, Lie87, NS89], blocks with nonequidistant solution values have been considered (like Runge–Kutta methods). The block structure of both families of correctors makes it possible to implement the PC method efficiently on a parallel computer system. Moreover, parallel computers also allow us to use local Richardson extrapolation for automatic stepsize control without additional *sequential* costs, because the "reference" solution used in the error estimate can be computed in parallel.

In applying PC methods, we may fix the number of iterations in advance $(\text{PE(CE)}^m$ mode with m usually 1 or 2), or we may iterate until the iterated values satisfy accuracy requirements. The first strategy was followed by Birta and Abou–Rabia [BAR87] and by Chu and Hamilton [CH87]. A disadvantage of this approach is that the stability regions usually are extremely small, unless the corrector is tuned to the particular PE(CE)^m mode employed. For example, the real stability boundaries of the "best" PECE methods constructed by Birta and Abou–Rabia (methods using the "null-weight predictor") range from $\beta_{\text{re}} = 0.576$ for blocksize 2 until $\beta_{\text{re}} = 0.078$ for blocksize 10. By using a number of free parameters in the corrector for improving stability, Chu and Hamilton [CH87]

succeeded in increasing the real stability boundary substantially. However, for a given blocksize, the order is of course reduced. Both [BAR87] and [CH87] restrict the stability considerations to the real axis.

In the second strategy, where the corrector is iterated until the iterated values satisfy our accuracy requirements, we are not only faced with the stability region of the corrector, but also with its *convergence* region. The cross-section of these regions may be interpreted as the stability region of the PC method. So far, the investigations have mainly been concerned with Runge–Kutta (RK) correctors. As it happens, the classical RK correctors of Gauss–Legendre or Radau type, have a high order of accuracy (superconvergence), they are unconditionally stable, and they possess very large convergence boundaries. Hence, RK-based PC methods are both highly accurate and highly stable. Moreover, by their one-step nature, RK correctors allow an easy and highly efficient stepsize strategy (provided that the predictor formula is also of one-step type). In [HS90] experiments are reported showing that the sequential costs of one-step PC methods based on the Gauss–Legendre corrector of order 10 are about half the sequential costs of the DOPRI8 code. The DOPRI8 code of Hairer–Nørsett–Wanner [HNW87] is based on the 13-stage, 8^{th}-order embedded RK method of Prince and Dormand [PD81], and is generally considered as one of the most efficient sequential codes (cf. [HNW87, p. 378]). We remark that by sacrificing the one-step nature of the predictor–corrector pair, the efficiency of parallel RK-based PC methods can be improved drastically, of course at the cost of a less easy implementation and stepsize strategy. A few first experiments were reported in [Lie87].

The LM-based and RK-based PC methods discussed above are very special examples of methods that fit into the large class of *general linear methods* introduced by Butcher in 1966. In this chapter, we shall try to find more efficient predictor–corrector pairs than constructed so far by looking in this class of general linear methods. In particular, we shall combine the multistep nature of the Birta and Abou–Rabia and Chu and Hamilton methods with the non-equidistant-solution-values property of RK methods. In fact, the methods of this chapter belong to the family of Block Runge–Kutta (BRK) correctors studied in [HS92], where a first few results for BRK-based PC methods can be found. Here, we shall pursue these investigations. In particular, we pay attention to the stability of the PC method, because the weak point of most block methods is their small stability region.

In §2.2, we specify a family of two-stage BRK correctors and we discuss the order of accuracy and their stability. §2.3 analyzes PC iteration of these BRK correctors and defines the convergence factors associated with the iteration process. The main results of this chapter can be found in §2.4 where a number of BRK correctors are constructed that combine high order of accuracy, fast convergence and sufficiently large stability boundaries. Finally, in §2.5, PC methods based on BRK pairs are compared with DOPRI8, showing that speed-up factors derived from sequential function calls range from 1.9 until 2.9. When implemented on a parallel computer system, these speed-up factors will decrease. However, in earlier experiments with iterated RK methods on the Alliant FX4, we found that in this type of methods, the loss due to synchronization of

the processors is about 10% (this is confirmed by experiments of Lie [Lie87]). Since the new methods are of the same type as iterated RK methods, we may expect that the speed-up factors based on sequential function calls will be reduced by only 10% when run on an Alliant.

2.2 Block Runge–Kutta methods

For the definition and analysis of the block Runge–Kutta (BRK) methods, it is convenient to introduce some notations. Firstly, we shall frequently use the componentwise notation for functions of vectors. For example, v^2 is understood to be the vector whose entries are the squares of the entries of v. Furthermore, $\mathbf{1}$ denotes the vector with unit entries, e_i the i^{th} unit vector whose entries vanish except for the i^{th} entry which equals 1, I_{dd} is the $d \times d$ identity matrix, and E_{mn} is the $m \times n$ matrix whose entries are zero except for its n^{th} column which equals $\mathbf{1}$. The dimension of $\mathbf{1}$ and e_i may change, but will always be clear from the context.

Our starting point is a method of the form

$$Y = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y)\,, \tag{2.2}$$

$$Y_n = (A^* \otimes I_{dd})Y_{n-1} + h(B^* \otimes I_{dd})F(Y_{n-1}) + h(C^* \otimes I_{dd})F(Y)\,, \tag{2.3}$$

$$y_n = y_{n-1} + h(b^{\mathrm{T}} \otimes I_{dd})F(Y_{n-1}) + h(c^{\mathrm{T}} \otimes I_{dd})F(Y_n), \quad n = 1, \ldots, N\,. \tag{2.4}$$

Here, the $s \times s$ matrices A, B, C, A^*, B^*, C^* and the s-dimensional vectors b and c contain the method parameters, h denotes the stepsize $t_n - t_{n-1}$, and $\otimes$ denotes the Kronecker product. Furthermore, Y and Y_n represent numerical approximations to the exact solution vectors $y(t_{n-1} + a_i h)$, where $a = (a_i)$, $i = 1, \ldots, s$, denotes the abscissa vector, and where for any vector $V = (V_i)$, $F(V)$ contains the derivative values $f(V_i)$. It is assumed that the components of a are distinct.

The formulas (2.2), (2.3) and (2.4) are respectively called the stage vector equation with s *internal* stages, the output formula with s *external* stages, and the step point formula. The *external* stages are all explicit, while the internal stages can be implicit or explicit. For example, if C has q zero rows and $r := s - q$ rows with nonzero entries, then there are q explicit stages and r fully implicit stages. The quantities Y, Y_n and y_n are respectively called the stage vector, the output vector and the step point value.

With respect to parallel implementation, methods of this type have been studied in [HS92], and were called Block Runge–Kutta (BRK) methods, because they can be obtained from conventional RK methods by replacing the scalar RK parameters by matrices and the stage values by blocks of stage values. Like RK methods, the stage values correspond to nonuniformly spaced points at the t-axis. In terms of the array notation used in [HS92], the method (2.2)–(2.4) can be represented as a two-stage BRK method with one explicit and one implicit (block) stage:

$$\begin{array}{c|cc} I & 0 & 0 \\ A & B & C \\ \hline A^* & B^* & C^* \end{array}\,. \tag{2.5}$$

In the determination of the parameter matrices in the stage vector equation (2.2), the order conditions (see §2.2.1) will play an important role, together with the requirement that the iteration method used for solving the stage vector equation is rapidly converging. In this chapter, we will solve the stage vector equation by predictor–corrector (PC) type iteration. The convergence of PC iteration is largely controlled by the "magnitude" of the matrix C, that is, convergence is better as C is smaller in some sense. For example, its spectral radius is often a first indicator of the potential convergence speed (see §2.3). In this connection, we should remark that strictly triangular matrices C lead to a zero spectral radius (in fact, the method is an explicit method, so that no iteration process is needed). However, such explicit methods approximate the components of Y by *extrapolation* formulas which are considerably less accurate than the *interpolation* formulas associated with *implicit* methods. Fortunately, it turns out that high accuracy and fast convergence often go together. Hence, if the stage vector Y has sufficient accuracy and stability, then the output formula (2.3) can be dropped (i.e. $A = A^*$, $B = B^*$, $C = C^*$, so that $Y_n = Y$).

In most of the BRK correctors constructed in this chapter, we do not use an output formula. However, we shall show in §2.4.3 that output formulas can be used for stabilizing the corrector.

The step point formula can be used to increase the order at the step points (superconvergence). This can be achieved by setting $b = 0$ and by identifying the components of c and the abscissa vector a with the quadrature weights and quadrature points of Gaussian quadrature formulas. The step point order is then one higher than the order of the output vector Y_n. Since in the methods considered in this chapter, the order of the output vector will be at most $s + 1$ or $s + 2$, there is, as far as order of accuracy is concerned, no need for basing the abscissa vector on quadrature formulas of the highest possible order (Gauss–Legendre formulas). This leads us to use abscissa vectors with $a_1 \neq 0$ and $a_s = 1$ which often simplifies the implementation (e.g. the Radau II points fit into this group).

Finally, we remark that (2.2)–(2.4) reduces to an RK method by setting $A = A^* = E_{ss}$, $B = B^* = 0$, $C = C^*$, $b = 0$ and $c^{\mathrm{T}} = e_s^{\mathrm{T}} C$ with C denoting the RK matrix of the collocation method defined by the abscissa vector a. We shall call this collocation method the *RK method associated with the abscissa vector a*. By identifying the BRK method (2.2)–(2.4) in the first step with such an RK method, we can avoid the problem of computing starting values, because we only need the initial value y_0, and not the whole starting vector Y_0.

2.2.1 Accuracy

Given the abscissa vector a, the conditions for p^{th}-order consistency of the stage vector equation (2.2) are given by (see, e.g., [HS92])

$$A\mathbb{1} = \mathbb{1}, \quad A(a - \mathbb{1})^j + jB(a - \mathbb{1})^{j-1} + jCa^{j-1} = a^j\,, \quad j = 1, \ldots, p\,.$$

We may write these order conditions in the form

$$A\mathbb{1} = \mathbb{1}, \quad AX_{sp} + BW_{sp} + CV_{sp} = U_{sp}\,, \tag{2.6}$$

$$U_{sp} := (\tfrac{1}{j}a^j),\ V_{sp} := (a^{j-1}),\ W_{sp} := ((a-\mathbb{1})^{j-1}),\ X_{sp} := (\tfrac{1}{j}(a-\mathbb{1})^j),\ \ j = 1,\ldots,p\,,$$

where the lower indices again refer to the number of rows and columns of the matrix. If (2.6) is satisfied, then p will be called the *internal stage order*. The vector of principal error constants associated with the stage vector equation is given by

$$E_{p+1} := \tfrac{1}{(p+1)!}\left(a^{p+1} - A(a-\mathbb{1})^{p+1} - (p+1)\left[\begin{array}{cc} B & C \end{array}\right]\left(\begin{array}{c} (a-\mathbb{1})^p \\ a^p \end{array}\right)\right). \tag{2.7}$$

Note that for $p = s$, $A = E_{ss}$ and $B = 0$, the stage vector equation reduces to the stage vector equation of the RK method with RK matrix $C = U_{ss}V_{ss}^{-1}$.

For the output formula (2.3) we proceed as follows. Imposing the localizing assumption, i.e., assuming that the components $Y_{n-1,i}$ are on the locally exact solution through the point (t_{n-1}, y_{n-1}), we may set $Y_{n-1} = y(\mathbb{1}\,t_{n-2} + ah)$ and, by virtue of (2.6), $Y = y(\mathbb{1}\,t_{n-1} + ah) + \mathcal{O}(h^{p+1})$. Hence,

$$\begin{aligned} Y_n &= (A^* \otimes I_{dd})Y_{n-1} + h(B^* \otimes I_{dd})F(Y_{n-1}) + h(C^* \otimes I_{dd})F(Y) \\ &= (A^* \otimes I_{dd})y(\mathbb{1}\,t_{n-2} + ah) + h(B^* \otimes I_{dd})y'(\mathbb{1}\,t_{n-2} + ah) \\ &\quad + h(C^* \otimes I_{dd})y'(\mathbb{1}\,t_{n-1} + ah) + \mathcal{O}(h^{p+2})\,. \end{aligned}$$

By Taylor expansion it can be shown that

$$Y_n = y(\mathbb{1}\,t_{n-1} + ah) + \mathcal{O}(h^{p+2}) + \mathcal{O}(h^{p^*+1})\,,$$

provided that

$$A^*\mathbb{1} = \mathbb{1}, \quad A^*X_{sp^*} + B^*W_{sp^*} + C^*V_{sp^*} = U_{sp^*}\,, \tag{2.8}$$

where the matrices X_{sp^*}, W_{sp^*}, V_{sp^*}, and U_{sp^*} are defined as in (2.6) with p replaced by p^*. Thus, the output vector Y_n has order $\min\{p+1, p^*\}$. This order will be called the *external stage order*, or briefly *the* stage order.

There are two error vectors associated with the output formula, viz.

$$\begin{aligned} E^*_{1,p+2} &:= C^*E_{p+1}, \\ E^*_{2,p^*+1} &:= \tfrac{1}{(p^*+1)!}\left(a^{p^*+1} - A^*(a-\mathbb{1})^{p^*+1} - (p^*+1)\left[\begin{array}{cc} B^* & C^* \end{array}\right]\left(\begin{array}{c} (a-\mathbb{1})^{p^*} \\ a^{p^*} \end{array}\right)\right), \end{aligned} \tag{2.9}$$

where E_{p+1} is defined by (2.7).

Finally, we consider the step point formula (2.4). It is possible to achieve order of consistency $2s$ for this formula, so that the order at the step points becomes $\min\{2s, p+2, p^*+1\}$. However, this may lead to rather large entries in b and c. Alternatively, we may use a zero b vector and identify c^{T} with the last row vector of the RK matrix associated with a, that is, $c^{\mathrm{T}} = e_s^{\mathrm{T}}U_{ss}V_{ss}^{-1}$. If p_{RK} denotes the order of the RK method, then we have order of accuracy $\min\{p_{\mathrm{RK}}, p+2, p^*+1\}$ at the step points. This will be called the step point order. We summarize the preceding discussion in the following theorem:

THEOREM 2.1 *If (2.6) and (2.8) are satisfied, then the BRK method (2.2)-(2.4) has stage order* $min\{p+1, p^*\}$ *and output vector errors given by (2.9). If, in addition,* $b = 0$, $c^{\mathrm{T}} = e_s^{\mathrm{T}} U_{ss} V_{ss}^{-1}$, *and if the abscissa vector* a *defines an RK method of order* p_{RK}, *then the step point order is given by* $\min\{p_{\mathrm{RK}}, p+2, p^*+1\}$.

2.2.2 Stability

In order to ensure stability for $h = 0$ (zero-stability), we shall require that A^* has $s-1$ eigenvalues inside the unit circle (since (2.8) prescribes that $A^*\mathbb{1} = \mathbb{1}$, A^* necessarily has one eigenvalue 1). Such matrices will be referred to as *zero-stable matrices.*

For $h > 0$, stability also depends on the other parameter matrices and on the abscissa vector a. With respect to the scalar test equation $y' = \lambda y$, where λ runs through the spectrum of the Jacobian matrix $\partial f/\partial y$, we obtain the recursion

$$Y_n = M(z)Y_{n-1}\,, \quad M(z) := A^* + zB^* + zC^*(I - zC)^{-1}(A + zB)\,, \quad z := \lambda h\,. \tag{2.10}$$

If we assume that the stability matrix $M(z)$ has s distinct eigenvalues, then we have that (cf. [Var62])

$$\|Y_n\|_2 \le \|M^n(z)\|_2 \|Y_0\|_2\,, \quad \|M^n(z)\|_2 \approx \nu(z)[\rho(M(z))]^n \quad \text{as} \quad n \to \infty\,, \tag{2.11}$$

where $\nu(z)$ is bounded by the condition number of the eigensystem of $M(z)$. This estimate suggests defining the stability region, and the real and imaginary stability intervals according to

$$\mathcal{S} := \{z : \rho(M(z)) < 1\}\,, \tag{2.12}$$

$$(-\beta_{\mathrm{re}}, 0) := \{z : z \in \mathcal{S}, z < 0\}\,, \quad (-\beta_{\mathrm{im}}, \beta_{\mathrm{im}}) := \{z : z \in \mathcal{S}, \mathrm{Re}(z) = 0, z \neq 0\}\,,$$

where $\rho(\cdot)$ denotes the spectral radius function. The quantities β_{re} and β_{im} are respectively called the *real* and the *imaginary stability boundary* of the BRK method. By (2.12) stability conditions of the type $h < \beta/\rho(\partial f/\partial y)$ are implied, so that we should require the method to have sufficiently large stability boundaries, say not less than 1. In addition, we should impose the condition that $\nu(z)$ is of moderate size, particularly for $z = 0$, because zero-stability implies $\rho(M(0)) = \rho(A^*) = 1$, so that

$$\|Y_n\|_2 \le \nu(0)\|Y_0\|_2 \quad \text{as} \quad n \to \infty\,. \tag{2.13}$$

If A^* is singular, then estimating an upper bound for $\nu(0)$ by means of the condition number of A^* is not possible. For example, this happens in the important case where $A^* = E_{ss}$ (in [HS92] BRK methods of this form were called BRK methods of *Adams type*). However, for such methods, $M^n(0) = [A^*]^n = E_{ss}$, hence $\|M^n(0)\|_2 = \|E_{ss}\|_2 = \sqrt{s}$, so that for $z = 0$, Adams-type BRK methods satisfy (2.11) with $\nu(0) = \sqrt{s}$.

2.3 The iteration scheme

We approximate the solution Y of (2.2) by successive iterates $Y^{(j)}$ satisfying the PC scheme (or fixed-point iteration scheme)

$$Y^{(j)} = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y^{(j-1)}), \qquad j = 1, \ldots, m;\ n \geq 1. \tag{2.14}$$

Evidently, if the iterates $Y^{(j)}$ satisfying (2.14) converge to a fixed vector V as $j \to \infty$, then $V = Y$. In actual computation, the number of iterations m is dynamically determined by requiring that the corrector equation is solved within a given tolerance (cf. §2.5). This iteration scheme has a high degree of parallelism, because the *sequential* costs of each iteration on s processors are independent of the number of implicit stages r.

For the predictor formula providing $Y^{(0)}$, we may take the explicit BRK method

$$Y^{(0)} = (A_0 \otimes I_{dd})Y_{n-1} + h(B_0 \otimes I_{dd})F(Y_{n-1}). \tag{2.15}$$

One option defines A_0 and B_0 according to

$$[\,A_0\ B_0\,] = [\,U_{s,2s-1}\ \mathbb{1}\,] \begin{bmatrix} X_{s,2s-1} & \mathbb{1} \\ W_{s,2s-1} & 0 \end{bmatrix}^{-1}, \tag{2.16}$$

where $X_{s,2s-1}$, $W_{s,2s-1}$ and $U_{s,2s-1}$ are defined as in (2.6). It is easily seen that (2.16) satisfies (2.6) for $A = A_0$, $B = B_0$, $C = 0$ and $p = 2s - 1$, so that (2.15)–(2.16) generate predictor values of order $2s - 1$ (provided that the stage order of the BRK method (2.2)–(2.4) is at least $2s - 1$). In fact, (2.16) is a Hermite integration formula generated by the abscissa vector a. In addition to the high orders of Hermite formulas, the error constants $\|E_{p+1}\|_\infty$ as defined in (2.7) are extremely small. In Table 2.1, this is illustrated for the Radau II abscissae. However, in spite of their high orders and relatively small error constants, Hermite predictor formulas have the drawback of extremely large coefficients in the parameter matrices A_0 and B_0, especially for larger values of s. This may cause considerable round-off errors unless sufficiently high arithmetic is used (we remark that to some extent, round-off can be suppressed by using shifted iterates $X^{(j)} := Y^{(j)} - \mathbb{1} \otimes y_{n-1}$ in an actual implementation (cf. [HW91, p. 128])).

An alternative to the Hermite predictor formula is offered by the Adams–Bashforth-type formula defined by

$$A_0 = E_{ss}, \quad B_0 = U_{ss}W_{ss}^{-1}, \tag{2.17}$$

which satisfies (2.6) for $A = E_{ss}$, $B = B_0$, $C = 0$ and $p = s$. Its error constants associated with the Radau II abscissae can be found in Table 2.1. From these figures it is clear that if arithmetic allows, we should use the Hermite predictors.

The method (2.14)–(2.15) will be called a PIBRK method (Parallel Iterated BRK method). The sequential costs of PIBRK methods depend on the structure of the parameter matrices. Therefore, we postpone a discussion of computational costs until the special cases developed in this chapter have been specified.

TABLE 2.1: *Error constants $\|E_{p+1}\|_\infty$ associated with Radau IIA abscissae for Hermite and Adams–Bashforth predictor formulas.*

Predictor	s = 2	s = 3	s = 4	s = 5
Hermite	0.12	0.0087	0.00034	0.0000081
Adams–Bashforth	0.33	0.13	0.041	0.010

For the convergence analysis of (2.14) we define the iteration error

$$\epsilon^{(j)} := Y^{(j)} - Y\,.$$

On substitution in (2.14), we obtain

$$\epsilon^{(j)} = h(C \otimes I_{dd})[F(Y^{(j-1)}) - F(Y)]\,.$$

This relation immediately leads to the estimate

$$\|\epsilon^{(j)}\| \leq hL\|C\|\,\|\epsilon^{(j-1)}\|\,,$$

where L denotes a Lipschitz constant on the right-hand side function f. Although this estimate has the advantage of being valid for the general IVP (2.1), it does not provide much information for selecting efficient corrector methods. Therefore, we resort to the familiar approach of approximating the IVP by a linear model. In this way, we obtain detailed information on the iteration process for the class of linear IVPs. Like the linear stability theory, this linear convergence theory turns out to be highly reliable for a large class of *non-linear* problems.

Assuming that the right-hand side function f is sufficiently smooth, we may write

$$F(U + \delta) - F(U) = J(U)\delta + \mathcal{O}(\delta^2)\,,$$

where $J(U)$ is an $sd \times sd$ block-diagonal matrix whose diagonal blocks consist of the Jacobian matrices $\partial f(U_i)/\partial y$, U_i being the components of U. On substitution, we straightforwardly derive the error recursion

$$\epsilon^{(j)} = Z\epsilon^{(j-1)} + \mathcal{O}(\epsilon^{(j-1)})^2,$$

where the matrix

$$Z = Z(hJ(Y)) := h(C \otimes I)J(Y) \tag{2.18}$$

controls the convergence of the iteration scheme. Assuming that higher-order terms can be neglected, the iteration error of the stage vector satisfies

$$\epsilon^{(j)} = [Z(hJ(Y))]^j\epsilon^{(0)} = h^j[(C \otimes I_{dd})J(Y)]^j\epsilon^{(0)}. \tag{2.19}$$

Thus, the iteration matrix C plays a crucial role in the convergence of the PC iteration process.

We shall define the (averaged) *convergence factor* for the scalar test equation $y' = \lambda y$. For this test equation, (2.19) reduces to

$$\epsilon^{(j)} = z^j C^j \epsilon^{(0)}, \quad z := \lambda h . \tag{2.20}$$

Hence,

$$\|\epsilon^{(m)}\|_\infty \leq |z|^m \|C^m\|_\infty \|\epsilon^{(0)}\|_\infty ,$$

so that, with respect to the maximum norm, the (averaged) convergence factor over m iterations is given by

$$\alpha(m, z) := |z| \sqrt[m]{\|C^m\|_\infty} . \tag{2.21}$$

The region of convergence in the complex z-plane is given by $\alpha(m, z) < 1$, that is, the open disk

$$\mathcal{C}_m := \{z : |z| < \gamma_m\}, \quad \gamma_m := \frac{1}{\sqrt[m]{\|C^m\|_\infty}} , \tag{2.22}$$

where γ_m may be considered as the *convergence boundary*. From (2.22) we deduce the stepsize condition $h < \gamma_m / \rho(\partial f / \partial y)$. Thus, large convergence boundaries relax the convergence condition and improve convergence at the same time.

In actual computation, one should satisfy both the convergence condition associated with (2.22) and the stability condition associated with (2.12), that is, the spectrum of the matrix $h\partial f/\partial y$ should be contained in the intersection of the stability region $\mathcal{S}$ and the convergence region $\mathcal{C}_m$ (here, we assume that the IVP is itself stable, so that the spectrum of $\partial f/\partial y$ is located in the left half-plane). As a consequence, there is no point in trying to construct correctors whose stability region is much larger than their region of convergence. However, it may be feasible to have correctors whose convergence region is much larger than their region of stability, because, as we just saw, large regions of convergence also improve convergence speed. Notice that strictly lower (or upper) triangular matrices C have zero convergence factors for $m \geq s + 2$. However, as already remarked, then the generating BRK corrector (2.2)–(2.4) is explicit and therefore has reduced accuracy.

2.4 Construction of BRK correctors

In all BRK correctors considered in this chapter, the abscissa vector a is identified with the Radau II points. We do not claim that these points are optimal, but it is likely that results based on Radau II points are indicative for other sets of abscissae.

In the construction of BRK correctors, we have to take into account: (i) the consistency conditions (2.6) and (2.8), (ii) the zero-stability and condition of the matrix A^*, (iii) the stability region, and (iv) the rate of convergence. These aspects will be characterized by the step point order, by the condition number $\kappa_\infty(A^*)$ of A^* (provided A^* is non-singular), the stability boundaries defined by (2.12), the condition number $\kappa_\infty(C)$ and the convergence boundaries defined in (2.22).

2.4.1 Adams-type correctors without output formulas

We start with the class of methods without output formula, that is, the output formula coincides with the stage vector equation, so that $A = A^*$, $B = B^*$ and $C = C^*$ (that is, there are no external stages). Within this class, zero-stability is automatically achieved by choosing the subclass of Adams methods, i.e. $A = E_{ss}$. One option for choosing the remaining matrices B and C is such that a high order of consistency is obtained. In our preliminary analysis of this type of methods we found that in general the convergence factors associated with the matrix C improve as the order of consistency increases. In particular, we observed that the entries in the upper part and in the lower right-hand corner of C are relatively small. This observation led us to consider BRK correctors of which the matrix C is of the form

$$C = \begin{bmatrix} 0 & 0 \\ \underline{C}_1 & \underline{C}_2 \end{bmatrix},$$

where $\underline{C}_1$ and $\underline{C}_2$, respectively, are an $r \times q$ and an $r \times r$ matrix with $q+r = s$. Evidently, such correctors have r implicit stages and q explicit stages. In particular, the matrix $\underline{C}_2$ determines the convergence of the PC iteration process.

We shall define the first q rows of the matrix B completely by consistency conditions. From (2.6) it follows that the first q stages are consistent of order s if they coincide with the first q rows of the matrix $U_{ss}W_{ss}^{-1}$. In fact, the resulting formulas are Adams–Bashforth formulas (cf. the Adams–Bashforth predictor formula (2.17)). Although these formulas are based on pure extrapolation, the extrapolation errors are relatively small, because they correspond to the first (and therefore smaller) components of the abscissa vector a.

The class of methods indicated above is defined by

$$A = E_{ss}\,, \qquad B = (U_{ss} - CV_{ss})W_{ss}^{-1}\,, \qquad C = \begin{bmatrix} 0 & 0 \\ \underline{C}_1 & \underline{C}_2 \end{bmatrix}, \tag{2.23}$$

$$A^* = A\,, \qquad B^* = B\,, \qquad C^* = C\,, \tag{2.24}$$

where the $r \times s$ matrix $\underline{C} := [\,\underline{C}_1 \;\; \underline{C}_2\,]$ is still free. This method is zero-stable (because A is zero-stable). Since (2.23) and (2.24) satisfy (2.6) for $p = s$ it follows from Theorem 2.1 that the stage order equals s. In the following subsections, a few options for choosing the matrix $\underline{C}$ will be discussed.

Adams–Bashforth–Moulton methods

The most simple option defines the implicit stages by imposing consistency conditions of highest possible order, that is, $\underline{C}$ is defined by the $r \times s$ matrix occurring in the lower right-hand corner of the $s \times 2s$ matrix

$$U_{s,2s} \begin{bmatrix} W_{s,2s} \\ V_{s,2s-1} \end{bmatrix}^{-1}. \tag{2.25}$$

The resulting BRK corrector defines the first q components of $Y_n = Y$ by (explicit) Adams–Bashforth-type formulas (of order s) and the last r components of Y_n by implicit

Adams-type formulas of highest possible order of consistency (i.e. order $2s$). In this respect, these implicit formulas resemble the conventional Adams–Moulton formulas. Therefore, we shall refer to these correctors as Adams–Bashforth–Moulton correctors (ABM correctors). The special methods arising for $q = 0$ and $q = r$ will be called Adams–Moulton (AM) correctors and Adams–Bashforth (AB) correctors, respectively.

We recall that the stage order of ABM correctors equals s. However, for AM correctors where no explicit stages occur ($q = 0$), the stage order becomes $2s$. For $q > 0$, Theorem 2.1 shows that the step point order can be raised to $s + 1$ by choosing in the step point formula $b = 0$ and $c^{\mathrm{T}} = e_s^{\mathrm{T}} U_{ss} V_{ss}^{-1}$. However, it turns out that for ABM correctors $e_s^{\mathrm{T}} B \approx 0$ and $e_s^{\mathrm{T}} C \approx e_s^{\mathrm{T}} U_{ss} V_{ss}^{-1}$. Hence, in practical applications, we achieve superconvergence at the step points by defining the step point formula simply by $y_n = (e_s^{\mathrm{T}} \otimes I_{dd}) Y_n$.

We computed the convergence and stability characteristics for a large number of ABM correctors. In the case of a zero imaginary stability boundary, we have also computed the value of β_{im}^* defined by the length of the imaginary interval where the spectral radius of the amplification matrix $M(z)$ is bounded by $1 + \epsilon$, $\epsilon > 0$. For sufficiently small values of ϵ, these values can be used as the "effective" imaginary stability boundary in practical computations. For a given value of r, it turns out that the stability boundaries decrease and the convergence boundaries increase with q. For each r ($2 \leq r \leq 5$), Table 2.2 presents the two cases where the stability boundaries β_{re} and β_{im}^* (with $\epsilon = 10^{-3}$) are both sufficiently large (say at least ≈ 1) while the convergence boundaries are maximal. A more extensive list including cases with smaller convergence and stability boundaries can be found in the appendix to this chapter. Notice that a large condition number for $\underline{C}_2$ implies relatively small convergence boundaries in the first few iterations.

Next, we discuss the sequential costs of the PIBRK method based on ABM correctors. Let us consider the following implementation of the PIBRK method:

$$
\begin{aligned}
\underline{Y}^{(0)} &= (\underline{A}_0 \otimes I_{dd}) Y_{n-1} + h(\underline{B}_0 \otimes I_{dd}) F_{n-1}^* \,, \\
\overline{Y}^{(j)} &= (\overline{A} \otimes I_{dd}) Y_{n-1} + h(\overline{B} \otimes I_{dd}) F_{n-1}^* \,, \\
\underline{Y}^{(j)} &= (\underline{A} \otimes I_{dd}) Y_{n-1} + h(\underline{B} \otimes I_{dd}) F_{n-1}^* \\
&\qquad + h(\underline{C}_1 \otimes I_{dd}) F(\overline{Y}^{(j-1)}) + h(\underline{C}_2 \otimes I_{dd}) F(\underline{Y}^{(j-1)}) \,, \\
Y_n &= \begin{pmatrix} \overline{Y}^{(m)} \\ \underline{Y}^{(m)} \end{pmatrix}, \qquad y_n = (e_s^{\mathrm{T}} \otimes I_{dd}) Y_n, \qquad F_n^* = F(Y^{(m-1)}) \,,
\end{aligned}
\tag{2.26}
$$

where $j = 1, \ldots, m$. Here, upper and lower bars refer to the first q and last r rows of a matrix, and the underlying matrices A, B and C are defined by (2.23), (2.24) and (2.25). Notice that the first q components of the iterates $Y^{(j)}$ do not depend on j.

It is easily verified that (2.26) does yield the a solution to the corrector method as $m \to \infty$ (provided that it converges). Assuming that $1 \leq q \leq r$, the sequential costs on r processors are $m + 1$ right-hand side evaluations, that is, the evaluation of $F(\overline{Y}^{(m)})$ plus the evaluation of the m right-hand side functions $F(\underline{Y}^{(j-1)})$. The evaluation of $F(\overline{Y}^{(m)})$ can be done in parallel with that of $F(\underline{Y}^{(0)})$, but this would require q additional

TABLE 2.2: *Characteristics for selected ABM correctors.*

$s = q + r$	order	β_{re}	β_{im}	β^*_{im}	$\kappa_\infty(\underline{C}_2)$	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
$3 = 1 + 2$	4	3.33	0	3.81	17	2.48	3.08	3.55	5.16	...	6.17
$4 = 2 + 2$	5	0.94	0	0.91	15	3.82	4.55	5.20	7.79	...	9.06
$4 = 1 + 3$	5	2.88	0	2.53	81	1.79	2.39	2.96	5.91	...	8.23
$5 = 2 + 3$	6	1.26	0	1.78	64	2.48	3.26	3.99	7.47	...	10.35
$5 = 1 + 4$	6	1.88	0	2.90	383	1.68	2.11	2.59	5.45	...	10.68
$6 = 2 + 4$	7	1.38	0.99	0.99	239	2.06	2.64	3.26	6.60	...	12.28
$5 = 0 + 5$	6	2.26	0.01	3.16	13853	1.33	1.93	2.26	4.50	...	10.80
$6 = 1 + 5$	7	0.92	1.13	1.13	2700	1.56	2.05	2.47	5.12	...	13.05

processors. However, if we apply local Richardson extrapolation for stepsize control and if we use additional processors for computing the "reference" solution, then these processors can also be used for evaluating $F(\overline{Y}^{(m)})$. Since the "reference" solution is computed with a double step, it is likely that there is some idle time, in spite of the fact that larger steps will require more iterations to solve the stage vector equation. Hence, in such a case, the total sequential costs per step are just m right-hand side evaluations.

Adams–Bashforth–Radau methods

A second option identifies the matrix $[\underline{C}_1\ \underline{C}_2]$ in (2.23) with the last r rows of the Radau IIA matrix $U_{ss}V_{ss}^{-1}$. Then the matrix B vanishes, so that the r implicit stages are determined by Radau formulas. The stage order and the step point order are the same as for ABM correctors, i.e. s and $s+1$, respectively. We shall call this corrector an Adams–Bashforth–Radau corrector (ABR corrector) because the first q components of Y_n are defined by Adams–Bashforth formulas and the last r components by Radau IIA formulas. For $r = 0$ the corrector reduces to the AB corrector and $r = s$ leads to the Radau IIA corrector. The PIBRK method generated by ABR correctors can be defined according to (2.26), so that the sequential costs are equal.

The analogue of Table 2.2 is given in Table 2.3 where we included the case $q = 0$ defining the pure Radau IIA corrector. A comparison of these selected methods with the corresponding ABM correctors reveals that ABR correctors have smaller convergence boundaries (particularly for larger m), but possess considerably larger stability boundaries. One may argue that the stability boundaries of the selected ABM correctors are sufficiently large for integrating non-stiff problems, so that the ABM correctors seem to be the more attractive ones. However, if the stage vector equation is not solved to convergence (for example, if the tolerance parameter in the stopping criterion is not sufficiently small), then we are faced with the fact that the stability region of the ABM method is much smaller than that of the ABR method. §2.5 will show that ABR is more efficient than ABM because of its better stability characteristics for small numbers of iterations.

TABLE 2.3: *Characteristics for selected ABR correctors.*

$s = q + r$	order	β_{re}	β_{im}	β^*_{im}	$\kappa_\infty(\underline{C}_2)$	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
$2 = 0 + 2$	3	∞	∞	∞	7	1.41	1.59	1.86	2.36	...	2.45
$3 = 1 + 2$	4	8.30	4.32	4.32	9	2.15	2.48	2.87	3.66	...	4.31
$4 = 2 + 2$	5	1.05	0	0.93	10	3.39	3.92	4.49	5.93	...	7.11
$3 = 0 + 3$	5	∞	∞	∞	18	1.41	1.82	2.21	3.03	...	3.64
$4 = 1 + 3$	5	17.18	0.00	9.02	24	1.81	2.32	2.62	4.08	...	4.94
$5 = 2 + 3$	6	1.97	0.02	1.89	27	2.45	3.08	3.47	5.80	...	7.03
$4 = 0 + 4$	7	∞	∞	∞	34	1.41	1.82	2.21	4.28	...	5.04
$5 = 1 + 4$	6	30.16	0.04	15.74	44	1.65	2.11	2.55	4.84	...	5.99
$6 = 2 + 4$	7	3.35	0	2.86	50	2.04	2.61	3.15	5.80	...	7.74
$5 = 0 + 5$	9	∞	∞	∞	55	1.41	1.82	2.21	4.44	...	6.29
$6 = 1 + 5$	7	47.80	0	24.92	69	1.57	2.02	2.44	4.70	...	6.87
$7 = 2 + 5$	8	5.23	0.07	4.57	79	1.84	2.36	2.85	5.40	...	8.39

2.4.2 Adams-type correctors with Radau output formula

By adding an output formula (that is, introducing external stages), it is possible to improve the stability of the corrector method. We shall illustrate this by using output formulas of Radau-type:

$$A^* = E_{ss}\,, \quad B^* = 0\,, \quad C^* = U_{ss}V_{ss}^{-1}\,. \tag{2.27}$$

If the stage order of Y is p, then Y_n has stage order $\min\{p+1, s\}$ and step point order $s+1$.

The stability matrix $M(z)$ associated with (2.23) and (2.27) is given by

$$M(z) = E_{ss} + zU_{ss}V_{ss}^{-1}M_{\text{Adams}}(z)\,, \tag{2.28}$$

where $M_{\text{Adams}}(z)$ is the stability matrix of (2.23)–(2.24). The large entries in $M_{\text{Adams}}(z)$ are responsible for the possibly poor stability of (2.23)–(2.24). Since the entries of the Radau matrix $U_{ss}V_{ss}^{-1}$ are rather small, it is likely that the large entries in $M_{\text{Adams}}(z)$ are neutralized, so that the stability region of $M(z)$ is improved. This is confirmed by the Tables 2.4–2.5.

In comparison with the Adams methods without output formula, the higher stability has to be paid for by the additional evaluation of $F(Y_{n-1})$. This can be concluded from the following implementation of the generated PIBRK method (cf. (2.26)):

$$\begin{aligned}
\underline{Y}^{(0)} &= (\underline{A}_0 \otimes I_{dd})Y_{n-1} + h(\underline{B}_0 \otimes I_{dd})F(Y_{n-1})\,,\\
\overline{Y}^{(j)} &= (\overline{A} \otimes I_{dd})Y_{n-1} + h(\overline{B} \otimes I_{dd})F(Y_{n-1})\,,\\
\underline{Y}^{(j)} &= (\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F(Y_{n-1}) + h(\underline{C} \otimes I_{dd})F(Y^{(j-1)})\,,\\
Y_n &= (E_{ss} \otimes I_{dd})Y_{n-1} + h(U_{ss}V_{ss}^{-1} \otimes I_{dd})F(Y^{(m-1)})\,,\\
y_n &= (e_s^{\mathrm{T}} \otimes I_{dd})Y_n\,,
\end{aligned} \tag{2.29}$$

TABLE 2.4: *ABM (+ Radau) correctors.*

Corrector	$s = q + r$	order	β_{re}	β_{im}	β^*_{im}
ABM	$6 = 4 + 2$	7	0.02	0.02	0.02
ABM + R	$6 = 4 + 2$	7	1.51	0.02	1.40
ABM	$6 = 3 + 3$	7	0.17	0.03	0.18
ABM + R	$6 = 3 + 3$	7	1.98	1.79	1.79
ABM	$7 = 4 + 3$	8	0.01	0.01	0.01
ABM + R	$7 = 4 + 3$	8	1.01	0.01	0.95
ABM	$7 = 3 + 4$	8	0.19	0.22	0.22
ABM + R	$7 = 3 + 4$	8	2.18	1.83	1.83
ABM	$7 = 2 + 5$	8	0.47	0.36	0.36
ABM + R	$7 = 2 + 5$	8	2.31	0.03	2.78

TABLE 2.5: *ABR (+ Radau) correctors.*

Corrector	$s = q + r$	order	β_{re}	β_{im}	β^*_{im}
ABR	$6 = 4 + 2$	7	0.01	0.01	0.02
ABR + R	$6 = 4 + 2$	7	1.52	0.01	1.41
ABR	$6 = 3 + 3$	7	0.18	0.04	0.19
ABR + R	$6 = 3 + 3$	7	1.70	0	1.81
ABR	$7 = 4 + 3$	8	0.01	0.01	0.01
ABR + R	$7 = 4 + 3$	8	0.98	0	0.97
ABR	$7 = 3 + 4$	8	0.27	0.06	0.28
ABR + R	$7 = 3 + 4$	8	1.90	0.01	2.08
ABR	$8 = 3 + 5$	9	0.40	0.01	0.43
ABR + R	$8 = 3 + 5$	9	2.27	0.01	2.54

where $j = 1, \ldots, m$. Note that the evaluation of $F(Y_{n-1})$ cannot be replaced by F^*_{n-1} as in (2.26), because then the effect of the stabilizing output formula is not taken into account. However, in the ABR case (where the m^{th} iteration is identical with the output formula), we may replace the last r components of $F(Y_{n-1})$ by those of F^*_{n-1}, without changing the corrector solution. Thus, with respect to the method (2.26), the additional costs are one right-hand side evaluation in the ABR case and two right-hand side evaluations in the ABM case. As before, if we have q additional processors at our disposal, then the total sequential costs per step can be reduced by one right-hand side evaluation (cf. the discussion of the method (2.26)).

The Tables 2.4–2.5 illustrate the stabilizing effect of adding a Radau output formula.

2.4.3 More general correctors

In the ABM and ABR correctors of §2.4.1, the first q (explicit) stages have order s, so that the resulting stage order can never exceed s. The stage order can easily be increased

TABLE 2.6: *Characteristics for selected modified ABR correctors.*

$s = q + r$	order	β_{re}	β_{im}	β^*_{im}	$\kappa_\infty(\underline{C}_2)$	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
$3 = 1 + 2$	5	4.15	0	1.44	12	2.33	2.72	3.12	4.14	...	4.98
$4 = 1 + 3$	6	7.16	0	2.41	34	1.83	2.37	2.86	4.85	...	5.97
$6 = 2 + 4$	8	0.99	0.02	1.17	64	2.05	2.63	3.20	6.04	...	8.73
$7 = 2 + 5$	9	1.42	0.01	1.74	107	1.85	2.38	2.89	5.66	...	9.55

TABLE 2.7: Δ*-values for (2.30) obtained by (2.26) with (AB, ABM) and (AB, ABR) PC pairs.*

PC pair	$s = q + r$	h^{-1}	$m = 1$	$m = 2$	$m = 3$	$m = 4$	...	$m = \infty$
(AB, ABM)	$6 = 2 + 4$	10	*	*	*	*	...	3.7
(AB, ABR)	$6 = 2 + 4$	10	*	*	2.6	3.7	...	4.2
(AB, ABM)	$6 = 2 + 4$	20	0.5	*	3.0	6.5	...	7.5
(AB, ABR)	$6 = 2 + 4$	20	0.5	4.5	5.9	6.5	...	6.9
(AB, ABM)	$6 = 2 + 4$	40	4.6	*	8.8	9.3	...	9.6
(AB, ABR)	$6 = 2 + 4$	40	4.6	7.2	9.0	9.2	...	9.3
(AB, ABM)	$6 = 2 + 4$	80	7.9	9.2	10.7	10.7	...	10.7
(AB, ABR)	$6 = 2 + 4$	80	7.9	9.4	12.0	11.5	...	11.5

by using a number of the zero entries occurring in the matrices A, B and C defined in (2.23). For example, adding to the ABR corrector, the $(s-1)^{st}$ column of $\overline{A}$ and the last column of $\underline{B}$ for satisfying additional consistency conditions, we obtain a corrector of order $s+1$. This corrector may be considered as a "minimal" modification of the ABR corrector and will be referred to as the modified ABR corrector. A drawback of these modified correctors is the rather large magnitude of the entries in the matrix $\overline{A}$, even in the case of this minimal modification. Using more zero entries for a further increase of the stage order leads to dramatically large entries, so that it does not seem feasible to use this approach for constructing correctors with stage order $\geq s+2$.

Since the matrix $\underline{C}$ of the modified ABR correctors is no longer defined by the Radau IIA formulas, the step point formula $y_n = (e_s^T \otimes I_{dd})Y_n$ does not have superconvergence at the step points. Nevertheless, in practical applications we do observe step point order $s+2$, because again it turns out that $e_s^T B \approx 0$ and $e_s^T C \approx e_s^T U_{ss} V_{ss}^{-1}$ (cf. the discussion in §2.4.1). Hence, the modified ABR method can be implemented according to (2.26), so that the sequential costs per step are the same as for the ABM and ABR methods.

The characteristics of a few modified ABR correctors are summarized in Table 2.6. A comparison with the corresponding ABR correctors of Table 2.3 reveals that the modification leads to comparable convergence boundaries and smaller stability boundaries. However, the stage order and (effective) step point order is raised by one. A detailed investigation of this promising family of methods will be subject of the next chapter.

2.5 Numerical experiments

Our numerical tests were performed using 15-digits arithmetic. The accuracies obtained are given by the number of correct digits Δ, defined by writing the maximum norm of the absolute error at the endpoint in the form $10^{-\Delta}$. The PIBRK method is implemented according to (2.26) with the PC pairs (AB, ABM) and (AB, ABR), where the correctors have orders 7 and 8, and are selected from the Tables 2.2–2.3. In view of the relatively high corrector orders and the 15-digits arithmetic, we did not use Hermite predictors.

First, we will make a mutual comparison between ABM and ABR correctors, using a constant number of iterations per step. For that purpose, we select the following well-known test problems (cf. [HNW87]), viz. the Fehlberg problem

$$\begin{array}{rcll} y_1' & = & 2ty_1 \log(\max\{y_2, 10^{-3}\}), & y_1(0) = 1, \\ y_2' & = & -2ty_2 \log(\max\{y_1, 10^{-3}\}), & y_2(0) = \mathrm{e}, \end{array} \qquad 0 \le t \le 5, \tag{2.30}$$

and the Euler problem

$$\begin{array}{rcll} y_1' & = & y_2 y_3, & y_1(0) = 0, \\ y_2' & = & -y_1 y_3, & y_2(0) = 1, \\ y_3' & = & -0.51 y_1 y_2, & y_3(0) = 1, \end{array} \qquad 0 \le t \le 20. \tag{2.31}$$

Secondly, in §2.5.2, we add a dynamic iteration strategy to the 8^{th}-order (AB, ABR) and we compare this code with three existing codes from the literature. The chapter is concluded with a performance evaluation of these four codes on the Brusselator problem [HW91, p. 381] which was transformed into an IVP for ODEs of dimension $d = 882$.

2.5.1 Comparison of ABM and ABR correctors

We applied the PC pairs (AB, ABM) and (AB, ABR) with $s = 2+4$. These correctors are both of order 7, require four processors, and are equally expensive. The Tables 2.7–2.8 present Δ-values for a few values of h and m (overflow is indicated by $*$). From these figures, we may conclude that the efficiency of the two methods is comparable in the case of convergence, but for larger stepsizes (AB, ABR) is more robust than (AB, ABM). This can be explained by the larger stability regions of the (AB, ABR) method.

2.5.2 Comparisons with DOPRI8, PIRK8 and PIRK10

Since (AB, ABR) pairs are more stable and therefore more robust, we restrict our considerations to this family of PC methods. In particular, we tested the $s = 2 + 5$ method. This parallel, 8^{th}-order (AB, ABR) method was compared with the 8(7) RK pair of Prince and Dormand [PD81] and the parallel PC methods based on Gauss–Legendre correctors of order 8 and 10. For the Dormand–Prince method we took the DOPRI8 implementation of Hairer, Nørsett and Wanner [HNW87], and for the Gauss–Legendre methods we used the four and five-processor one-step codes PIRK8 and PIRK10 developed in [HS90].

TABLE 2.8: *Δ-values for (2.31) obtained by (2.26) with (AB, ABM) and (AB, ABR) PC pairs.*

PC pair	$s = q + r$	h^{-1}	$m = 1$	$m = 2$	$m = 3$	$m = 4$	...	$m = \infty$
(AB, ABM)	$6 = 2 + 4$	1	*	*	*	*	...	4.6
(AB, ABR)	$6 = 2 + 4$	1	*	*	3.2	3.9	...	4.9
(AB, ABM)	$6 = 2 + 4$	2	*	*	2.2	6.4	...	6.5
(AB, ABR)	$6 = 2 + 4$	2	*	4.6	5.4	6.6	...	6.4
(AB, ABM)	$6 = 2 + 4$	4	4.4	*	8.0	8.4	...	8.4
(AB, ABR)	$6 = 2 + 4$	4	4.4	7.7	8.1	8.3	...	8.3
(AB, ABM)	$6 = 2 + 4$	8	7.2	9.1	10.5	10.6	...	10.6
(AB, ABR)	$6 = 2 + 4$	8	7.1	10.2	10.4	10.4	...	10.4

The (AB, ABR) method was equipped with a dynamic iteration strategy based on the requirement that the step point component of the residue left on substitution of the j^{th} iterate into the stage vector equation should be less than a tolerance parameter TOL, i.e.

$$\|(e_s^{\mathrm{T}} \otimes I_{dd}) R^{(j)}\| \leq \mathrm{TOL}\,,$$

where

$$R^{(j)} := Y^{(j)} - (E_{ss} \otimes I_{dd}) Y_{n-1} - h(B \otimes I_{dd}) F(Y_{n-1}) - h(C \otimes I_{dd}) F(Y^{(j)})\,.$$

According to (2.26) we may write

$$R^{(j)} = Y^{(j)} - Y^{(j+1)} - h(B \otimes I_{dd})(F(Y_{n-1}) - F^*_{n-1})\,.$$

Clearly, the error caused by the iteration process should be smaller than the local truncation error. This is achieved by requiring TOL to be a factor δ less than the local error. In our experiments, we shall estimate the local error at t_{n-1} by $\|(e_s^{\mathrm{T}} \otimes I_{dd})(Y_{n-1} - Y^{(0)}_{n-1})\|$, where $Y^{(0)}_{n-1}$ denotes the prediction in the preceding step. Using the maximum norm and observing that $(e_s^{\mathrm{T}} \otimes I_{dd})(F(Y_{n-1}) - F^*_{n-1})$ vanishes in the case of ABR correctors, we are led to the stopping criterion

$$\|(e_s^{\mathrm{T}} \otimes I_{dd})(Y^{(j)} - Y^{(j+1)})\|_\infty \leq \delta \|(e_s^{\mathrm{T}} \otimes I_{dd})(Y_{n-1} - Y^{(0)}_{n-1})\|_\infty\,. \tag{2.32}$$

Below, the resulting implementation will be referred to as the ABR8 code (we did not yet implement a stepsize strategy, so that the results produced by this code may be improved when this facility is included).

As a third test problem, we take the Brusselator problem (see [HNW87, p. 381]), defined by

$$\begin{aligned} \tfrac{\partial u}{\partial t} &= 1 + u^2 v - 4.4u + \alpha(\tfrac{\partial^2 u}{\partial x^2} + \tfrac{\partial^2 u}{\partial y^2})\,, \\ \tfrac{\partial v}{\partial t} &= 3.4u - u^2 v + \alpha(\tfrac{\partial^2 v}{\partial x^2} + \tfrac{\partial^2 v}{\partial y^2})\,, \end{aligned} \qquad 0 \leq x \leq 1\,, \qquad 0 \leq t \leq 23.5\,, \tag{2.33}$$

TABLE 2.9: *Number of sequential right-hand side evaluations for the Fehlberg problem (2.30).*

Method	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$	$\Delta = 11$	m
DOPRI8	595	759	963	1227	1574	1990	2503	2.3
PIRK8	379	495	623	786	978	1383	1874	1.5
PIRK10	327	388	490	704	884	977	1078	1.2
ABR8	240	335	430	532	689	846	1067	1.0

TABLE 2.10: *Number of sequential right-hand side evaluations for the Euler problem (2.31).*

Method	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$	$\Delta = 11$	m
DOPRI8	415	576	728	898	1133	1422	1817	2.9
PIRK8	294	381	534	728	961	1172	1746	2.3
PIRK10	252	297	357	426	580	730	920	1.5
ABR8	160	192	223	293	379	506	643	1.0

supplemented with homogeneous Neumann boundary conditions and the initial conditions

$$\begin{aligned} u(t=0,x,y) &= 0.5+y\,, \\ v(t=0,x,y) &= 1+5x\,. \end{aligned} \tag{2.34}$$

Furthermore, $\alpha = 2 \cdot 10^{-3}$ and N (the number of equidistant points in the spatial direction) is set to 21, resulting in an ODE-system of dimension 882.

The Tables 2.9–2.11 show results for the various methods (for the first two test problems, the results for DOPRI8, PIRK8, and PIRK10 were taken from [HS90]). In the ABR8 code, δ is set to 10^{-4}. To facilitate a comparison, the listed numbers of sequential right-hand side evaluations have been obtained by interpolation to arrive at integer values of Δ. Furthermore, we list the (averaged) factor by which the existing codes are more expensive than ABR8 (this factor is denoted by m). These tables clearly show that ABR8 is the most efficient solver. We see that the speed-up factor of ABR8 with respect to the 8^{th}-order code DOPRI8 (to be considered as one of the most efficient sequential codes) ranges from 1.9 until 2.9. For the four-processor PIRK8 code of order 8, and the five-processor 10^{th}-order code PIRK10 this factor is in the range 1.3–2.3 and 1.1–1.5.

2.6 Concluding remarks

The search for efficient parallel PC methods reported in this chapter has resulted in several fastly converging and sufficiently stable PC pairs. With respect to the fully automatic code DOPRI8, the averaged speed-up factor of the fixed stepsize, five-processor ABR8 code ranges from 1.9–2.9. The efficiency of this code can be improved by including

TABLE 2.11: *Number of sequential right-hand side evaluations for the Brusselator problem (2.33)–(2.34).*

Method	$\Delta = 5$	$\Delta = 6$	$\Delta = 7$	$\Delta = 8$	$\Delta = 9$	$\Delta = 10$	$\Delta = 11$	m
DOPRI8	1594	2376	2908	3570	4532	5985	7856	1.9
PIRK8	1161	1579	1950	2462	3225	4190	5448	1.3
PIRK10	901	1362	1824	2148	2673	3265	3989	1.1
ABR8	591	1169	1747	2237	2624	3288	3953	1.0

a stepsize strategy. If the local error estimate is based on local Richardson extrapolation where the "reference" solution is computed in parallel on an additional set of processors, then these processors can also be used for saving one function call per step (see the discussion of the scheme (2.26)). In the numerical examples of this chapter, this would increase the speed-up factor by about 20%.

Appendix

Extension of Table 2.2

$s = q + r$	order	β_{re}	β_{im}	β^*_{im}	$\kappa_\infty(C_2)$	γ_2	γ_3	γ_4	γ_{10}	γ_∞
$2 = 2 + 0$	3	0.91	*	0.29	-	∞	∞	∞	∞	∞
$3 = 3 + 0$	4	0.59	0.60	0.61	-	∞	∞	∞	∞	∞
$4 = 4 + 0$	5	0.48	*	0.44	-	∞	∞	∞	∞	∞
$5 = 5 + 0$	6	0.44	*	0.44	-	∞	∞	∞	∞	∞
$6 = 6 + 0$	7	0.41	*	0.42	-	∞	∞	∞	∞	∞
$7 = 7 + 0$	8	0.40	*	0.40	-	∞	∞	∞	∞	∞
$8 = 8 + 0$	9	0.39	*	0.39	-	∞	∞	∞	∞	∞
$2 = 1 + 1$	3	2.60	1.64	1.65	1.00	4.44	4.44	4.44	4.44	4.44
$3 = 2 + 1$	4	0.54	0.65	0.65	1.00	9.18	9.18	9.18	9.18	9.18
$4 = 3 + 1$	5	0.11	*	0.12	1.00	16.06	16.06	16.06	16.06	16.06
$5 = 4 + 1$	6	*	*	*	1.00	25.02	25.02	25.02	25.02	25.02
$6 = 5 + 1$	7	*	*	*	1.00	36.00	36.00	36.00	36.00	36.00
$7 = 6 + 1$	8	*	*	*	1.00	49.00	49.00	49.00	49.00	49.00
$8 = 7 + 1$	9	*	*	*	1.00	64.00	64.00	64.00	64.00	64.00
$2 = 0 + 2$	3	7.50	*	1.02	27.02	1.59	2.09	2.48	4.09	4.85
$3 = 1 + 2$	4	3.33	*	3.81	17.17	2.48	3.08	3.55	5.16	6.17
$4 = 2 + 2$	5	0.94	*	0.91	15.25	3.82	4.55	5.20	7.79	9.06
$5 = 3 + 2$	6	0.13	*	0.13	14.38	5.53	6.50	7.40	11.30	12.89
$6 = 4 + 2$	7	*	*	*	13.86	7.59	8.88	10.10	15.58	17.58
$7 = 5 + 2$	8	*	*	*	13.51	10.03	11.71	13.29	20.60	23.12
$8 = 6 + 2$	9	*	*	*	13.25	12.84	14.96	16.97	26.33	29.48
$3 = 0 + 3$	4	5.53	*	3.67	157.29	1.30	1.78	2.24	4.78	6.36
$4 = 1 + 3$	5	2.88	*	2.53	81.27	1.79	2.39	2.96	5.91	8.23
$5 = 2 + 3$	6	1.26	*	1.78	63.66	2.48	3.26	3.99	7.47	10.36
$6 = 3 + 3$	7	0.17	*	0.18	55.63	3.32	4.34	5.24	9.53	13.28
$7 = 4 + 3$	8	*	*	*	51.05	4.33	5.61	6.67	12.03	16.83
$8 = 5 + 3$	9	*	*	*	48.08	5.49	7.08	8.32	14.95	20.92
$4 = 0 + 4$	5	3.91	*	6.31	1109.38	1.53	1.77	2.14	4.61	8.38
$5 = 1 + 4$	6	1.88	*	2.90	383.04	1.68	2.11	2.59	5.45	10.68
$6 = 2 + 4$	7	1.38	0.99	0.99	238.64	2.06	2.64	3.26	6.60	12.28
$7 = 3 + 4$	8	0.19	0.22	0.22	180.61	2.57	3.31	4.07	8.01	14.71
$8 = 4 + 4$	9	*	*	*	150.77	3.17	4.10	5.01	9.67	17.67
$5 = 0 + 5$	6	2.26	*	3.16	1.39 E4	1.33	1.93	2.26	4.50	10.80
$6 = 1 + 5$	7	0.92	1.13	1.13	2684.02	1.56	2.05	2.47	5.12	13.05
$7 = 2 + 5$	8	0.47	0.36	0.36	1131.09	1.85	2.39	2.90	6.03	14.32
$8 = 3 + 5$	9	*	*	0.10	664.12	2.20	2.84	3.46	7.12	16.35
$6 = 0 + 6$	7	1.00	*	1.14	2.06 E5	1.12	1.70	2.23	4.50	13.07
$7 = 1 + 6$	8	0.34	0.36	0.36	2.37 E5	1.45	1.94	2.41	4.92	15.17
$8 = 2 + 6$	9	0.11	*	0.11	6762.53	1.73	2.22	2.72	5.61	16.60
$7 = 0 + 7$	8	0.35	0.36	0.37	3.62 E6	0.79	1.87	2.12	4.55	15.17
$8 = 1 + 7$	9	0.10	*	0.11	3.76 E5	1.33	1.96	2.32	4.82	17.33
$8 = 0 + 8$	9	0.10	*	0.11	8.21 E7	0.50	1.66	2.21	4.55	17.32

Extension of Table 2.3

$s = q + r$	order	β_{re}	β_{im}	β^*_{im}	$\kappa_\infty(C_2)$	γ_2	γ_3	γ_4	γ_{10}	γ_∞
$2 = 2 + 0$	3	0.91	*	0.29	-	∞	∞	∞	∞	∞
$3 = 3 + 0$	4	0.59	0.60	0.61	-	∞	∞	∞	∞	∞
$4 = 4 + 0$	5	0.48	*	0.44	-	∞	∞	∞	∞	∞
$5 = 5 + 0$	6	0.44	*	0.44	-	∞	∞	∞	∞	∞
$6 = 6 + 0$	7	0.41	*	0.42	-	∞	∞	∞	∞	∞
$7 = 7 + 0$	8	0.40	*	0.40	-	∞	∞	∞	∞	∞
$8 = 8 + 0$	9	0.39	*	0.39	-	∞	∞	∞	∞	∞
$2 = 1 + 1$	3	3.00	1.49	1.51	1.00	4.00	4.00	4.00	4.00	4.00
$3 = 2 + 1$	4	0.54	0.64	0.65	1.00	9.00	9.00	9.00	9.00	9.00
$4 = 3 + 1$	5	0.11	*	0.12	1.00	16.00	16.00	16.00	16.00	16.00
$5 = 4 + 1$	6	*	*	*	1.00	25.00	25.00	25.00	25.00	25.00
$6 = 5 + 1$	7	*	*	*	1.00	36.00	36.00	36.00	36.00	36.00
$7 = 6 + 1$	8	*	*	*	1.00	49.00	49.00	49.00	49.00	49.00
$8 = 7 + 1$	9	*	*	*	1.00	64.00	64.00	64.00	64.00	64.00
$2 = 0 + 2$	3	∞	∞	∞	7.00	1.41	1.59	1.86	2.36	2.45
$3 = 1 + 2$	4	8.30	4.32	4.32	9.34	2.15	2.48	2.87	3.66	4.31
$4 = 2 + 2$	5	1.05	*	0.93	10.25	3.39	3.92	4.49	5.93	7.11
$5 = 3 + 2$	6	0.13	*	0.13	10.70	5.03	5.81	6.63	8.98	10.75
$6 = 4 + 2$	7	*	*	*	10.94	7.04	8.14	9.26	12.78	15.21
$7 = 5 + 2$	8	*	*	*	11.09	9.42	10.89	12.38	17.31	20.48
$8 = 6 + 2$	9	*	*	*	11.19	12.16	14.06	15.98	22.56	26.57
$3 = 0 + 3$	4	∞	∞	∞	18.06	1.41	1.82	2.21	3.03	3.64
$4 = 1 + 3$	5	17.18	*	9.02	23.82	1.81	2.32	2.62	4.08	4.94
$5 = 2 + 3$	6	1.97	*	1.89	26.93	2.45	3.08	3.47	5.80	7.03
$6 = 3 + 3$	7	0.18	*	0.19	28.75	3.28	4.05	4.60	8.00	9.68
$7 = 4 + 3$	8	*	*	*	29.89	4.26	5.24	5.95	10.62	12.87
$8 = 5 + 3$	9	*	*	*	30.65	5.41	6.62	7.52	13.63	16.56
$4 = 0 + 4$	5	∞	∞	∞	34.19	1.41	1.82	2.21	4.28	5.04
$5 = 1 + 4$	6	30.16	*	15.74	43.75	1.65	2.11	2.55	4.84	5.99
$6 = 2 + 4$	7	3.35	*	2.86	49.85	2.04	2.61	3.15	5.80	7.74
$7 = 3 + 4$	8	0.27	*	0.28	53.85	2.54	3.24	3.91	7.11	9.96
$8 = 4 + 4$	9	*	*	*	56.59	3.14	4.00	4.77	8.69	12.59
$5 = 0 + 5$	6	∞	∞	∞	55.38	1.41	1.82	2.21	4.44	6.29
$6 = 1 + 5$	7	47.80	*	24.92	68.93	1.57	2.02	2.44	4.70	6.87
$7 = 2 + 5$	8	5.23	*	4.57	78.48	1.84	2.36	2.85	5.40	8.39
$8 = 3 + 5$	9	0.40	*	0.43	85.26	2.19	2.80	3.38	6.35	10.33
$6 = 0 + 6$	7	∞	∞	∞	81.63	1.41	1.82	2.21	4.50	7.66
$7 = 1 + 6$	8	70.66	*	37.01	99.26	1.53	1.96	2.39	4.73	7.96
$8 = 2 + 6$	9	7.61	*	7.29	112.56	1.73	2.22	2.69	5.22	9.32
$7 = 0 + 7$	8	∞	∞	∞	112.94	1.41	1.82	2.21	4.52	8.94
$8 = 1 + 7$	9	99.27	*	52.43	134.71	1.50	1.93	2.34	4.71	8.89
$8 = 0 + 8$	9	∞	∞	∞	149.32	1.41	1.82	2.21	4.53	10.30

Chapter 3

A special class of parallel predictor–corrector methods

Abstract The so-called Adams–Bashforth–Radau (ABR) methods were proposed in Chapter 2. An ABR method is a high-order parallel predictor–corrector method for solving non-stiff initial value problems, based on a combination of Adams–Bashforth and Radau formulas. Comparison of ABR with the famous sequential 8(7) Runge–Kutta method of Dormand and Prince showed speed-up factors of about 2.7. In this chapter, we improve the ABR methods by making them more accurate without any additional costs. This improved version increases the speed-up factor on the average to 3.1.

3.1 Introduction

We shall consider predictor–corrector methods (PC methods) for solving on parallel computers the (non-stiff) initial value problem

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}}. \tag{3.1}$$

In Chapter 2 a class of parallel PC methods has been proposed, including the Adams–Bashforth–Radau (ABR) methods. These methods showed a speed-up factor of about 2.7 compared to DOPRI8. The DOPRI8 code by Hairer-Nørsett-Wanner [HNW93] is an implementation of the 13-stage, 8^{th}-order embedded Runge–Kutta method of Dormand and Prince, and is generally accepted as one of the best sequential codes. In this chapter, we improve the ABR methods by increasing the order by 1. The convergence and stability characteristics turn out to be even slightly better than those of ABR, while the sequential costs and the number of processors are (almost) the same.

The outline of the chapter is as follows. In §3.2 we specify a subclass of the large class of General Linear Methods, introduced by Butcher in 1966, and describe how methods that fall into this class can be compared by means of accuracy, stability and convergence. §3.3 briefly describes the ABR methods proposed in Chapter 2. In §3.4 we propose a more accurate variant of ABR. How this variant can be implemented without any additional costs compared to ABR is presented in §3.5. Finally, in §3.6, numerical experiments will show that this new variant indeed performs better than ABR.

3.2 A subclass of the class of general linear methods

In the following, the vector with unit entries is denoted by $\mathbb{1}$, the i^{th} canonical basis vector by e_i, and the $d \times d$ identity matrix by I_{dd}. Furthermore, O_{mn} is the $m \times n$ zero

matrix and E_{mn} is the $m \times n$ matrix whose entries are zero, except for its n^{th} column which equals $\mathbb{1}$. If v is a vector, v^j stands for the vector whose entries are the j^{th} powers of the entries of v.

To solve (3.1) we use methods of the form

$$Y_n = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y_n), \tag{3.2}$$

$$y_n = y_{n-1} + h(c^{\mathrm{T}} \otimes I_{dd})F(Y_n). \tag{3.3}$$

This type of methods falls into the class of General Linear Methods introduced by Butcher (see [But87]). Here the $s \times s$ matrices A, B, C and the s-dimensional vector c^{T} contain the method parameters, h denotes the stepsize $t_n - t_{n-1}$ and $\otimes$ denotes the Kronecker product. Y_n is the so called *stage vector* which represents numerical approximations to the exact solution vectors $y(t_{n-1} + a_i h)$, where the s-dimensional vector $a = (a_i)$ denotes the abscissa vector. Hence Y_n is an sd-dimensional vector. In this chapter, we restrict ourselves to the case where the components of a are the Radau IIA collocation points. For any vector $V = (V_i)$, $F(V)$ contains the derivative values $f(V_i)$. Formulas (3.2) and (3.3) are respectively called the *stage vector equation* and the *step point formula.* Considering (3.2) as the correction equation we solve this equation by applying the PC scheme

$$Y_n^{(0)} = (A_0 \otimes I_{dd})Y_{n-1} + h(B_0 \otimes I_{dd})F(Y_{n-1}), \tag{3.4}$$

$$Y_n^{(j)} = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y_n^{(j-1)}), \qquad j = 1, \ldots, m, \tag{3.5}$$

$$Y_n = Y_n^{(m)}.$$

Next we describe how accuracy, stability and convergence of the PC scheme can be defined in terms of A, B, C and c.

3.2.1 Accuracy

The conditions for p^{th}-order consistency of the stage vector equation (3.2) are given by (see, e.g. [HS92])

$$A\mathbb{1} = \mathbb{1}, \quad AX_{sp} + BW_{sp} + CV_{sp} = U_{sp}, \tag{3.6}$$

where the $s \times p$ matrices U_{sp}, V_{sp}, W_{sp} and V_{sp} are defined by

$$U_{sp} := (\tfrac{1}{j}a^j), \quad V_{sp} := (a^{j-1}), \quad W_{sp} := ((a - \mathbb{1})^{j-1}), \quad X_{sp} := (\tfrac{1}{j}(a - \mathbb{1})^j), \quad \text{for } j = 1, \ldots, p.$$

If (3.6) is satisfied, then p will be called the *stage order.*

Note that, for $A = E_{ss}, B = O_{ss}$, and C fulfilling (3.6) with $p = s$, (3.2) reduces to the s-stage Radau IIA method.

In this chapter, we use a step point formula that coincides with the formula for the s^{th} stage of the Radau IIA method: $c_s^{\mathrm{T}} = e_s^{\mathrm{T}} U_{ss} V_{ss}^{-1}$. We will refer to this formula as

the *Radau IIA step point formula.* It can be shown (see Chapter 2) that for this case, the order of y_n (the so called *step point order*) equals $\min\{2s-1, p+1\}$, where p again denotes the stage order.

3.2.2 Stability

With respect to the scalar test equation $y' = \lambda y$, where λ runs through the spectrum of the Jacobian $\frac{\partial f(y)}{\partial y}$, we obtain for (3.2) the recursion

$$Y_n = M(z)Y_{n-1}\ ,\ M(z) := (I - zC)^{-1}(A + zB)\ ,\ z := \lambda h.$$

We define the *stability region* and the *real* and *imaginary stability intervals* according to

$$\begin{aligned} \mathcal{S} &:= \{z \in \mathbb{C} \mid \rho(M(z)) < 1\}, \\ (-\beta_{\mathrm{re}}, 0) &:= \{z \in \mathbb{C} \mid z \in \mathcal{S} \wedge z < 0\}, \\ (-\beta_{\mathrm{im}}, \beta_{\mathrm{im}}) &:= \{z \in \mathbb{C} \mid z \in \mathcal{S} \wedge \mathrm{Re}(z) = 0 \wedge z \neq 0\}, \end{aligned}$$

respectively, where $\rho(\cdot)$ denotes the spectral radius function. β_{re} and β_{im} are called the *real* and *imaginary stability boundary*, respectively.

For many methods that we consider in the next sections, it turns out that $\beta_{\mathrm{im}} = 0$. To circumvent the numerical uncertainty we also computed the *practical imaginary stability interval* defined by $(-\beta^*_{\mathrm{im}}, \beta^*_{\mathrm{im}}) := \{z \in \mathbb{C} \mid \rho(M(z)) < 1 + 10^{-3} \wedge \mathrm{Re}(z) = 0 \wedge z \neq 0\}$. In practical computations, β^*_{im} can be safely used as the imaginary stability boundary.

3.2.3 Convergence

For the convergence analysis of (3.5) we define the iteration error

$$\epsilon^{(j)} := Y_n^{(j)} - Y_n\ .$$

Application to the scalar test equation $y' = \lambda y$ and substitution in (3.5) yield

$$\epsilon^{(j)} := z^j C^j \epsilon^{(0)}\ ,\ z := \lambda h\ ,$$

and consequently

$$\|\epsilon^{(m)}\|_\infty \leq |z|^m \|C^m\|_\infty \|\epsilon^{(0)}\|_\infty\ .$$

This leads us to defining the *region of convergence* by

$$\mathcal{C}_m := \{z \in \mathbb{C} \mid |z| < \gamma_m\}\ ,\ \gamma_m := \frac{1}{\sqrt[m]{\|C^m\|_\infty}}\ ,$$

where γ_m may be considered as the convergence boundary.

3.3 Adams–Bashforth–Radau methods

Let us write the matrix C in the form

$$C = \begin{bmatrix} \overline{C}_1 & \overline{C}_2 \\ \underline{C}_1 & \underline{C}_2 \end{bmatrix},$$

where $\overline{C}_1$ and $\underline{C}_2$ are square matrices of size $q \times q$ and $r \times r$ respectively ($q + r = s$). From now on, upper and under bars refer to the first q and last r rows of an array.

Our first examination of methods of type (3.2) led to the observation that the convergence factors γ_m become larger as the order of consistency increases. In particular, we observed that the entries of $\underline{C}_2$ are relatively small. So ideally we would like to iterate solely with $\underline{C}_2$ and therefore we considered methods with $\overline{C}_1 = O_{qq}$ and $\overline{C}_2 = O_{qr}$. Thus the first q stages become explicit while the remaining r stages are solved by an iteration process that is determined by a 'small' matrix $\underline{C}_2$. The method can now be viewed as an r-processor method, since the iteration process is only invoked on r implicit stages, which can be evaluated in parallel.

If we choose $\underline{B} = O_{rs}$ and define the matrices $\overline{B}$, $\underline{C}_1$ and $\underline{C}_2$ by order conditions, while A is identified with the matrix E_{ss}, we see that both the q explicit and the r implicit stages are given order s. This method was called Adams–Bashforth–Radau (ABR) in Chapter 2.

In order to get reasonably large stability intervals, the number of implicit stages has to exceed the number of explicit stages ($r > q$). The characteristics of a few ABR methods are listed in Table 3.1.

For the predictor matrices A_0 and B_0 we can take $A_0 = E_{ss}$ and $B_0 = U_{ss}W_{ss}^{-1}$, i.e. B_0 is defined by order conditions. In Chapter 2 we referred to this predictor as the Adams–Bashforth (AB) predictor. Note that the first q rows of B_0 now coincide with $\overline{B}$. Hence for the first q stages we do not apply a corrector anymore after the prediction.

Here and in the following we assume that the costs of an algorithm are mainly determined by the number of right-hand side evaluations (denoted shortly by f-evaluations).

Since f-evaluations of different stage vector components can be done in parallel, the costs of ABR on r processors per time step are m sequential f-evaluations for the corrector and, provided that $q \leq r$, 2 sequential f-evaluations for the predictor. If we apply an economization by replacing $F(Y_{n-1})$ in (3.4) and (3.5) by

$$F_{n-1}^{*} := \begin{pmatrix} F(\overline{Y}_{n-1}^{(0)}) \\ F(\underline{Y}_{n-1}^{(m-1)}) \end{pmatrix},$$

then the sequential costs are reduced to $m + 1$ f-evaluations per time step.

3.4 Improved Adams–Bashforth–Radau methods

For ABR methods in every row $s + 1$ elements in the matrices A, B and C are determined by order conditions. Consequently, these methods have stage order s. In

TABLE 3.1: *Characteristics for selected ABR correctors.*

$s = q + r$	stage order	order	β_{re}	β^*_{im}	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
$5 = 2 + 3$	5	6	1.97	1.89	2.45	3.08	3.47	5.80	...	7.03
$6 = 2 + 4$	6	7	3.35	2.86	2.04	2.61	3.15	5.80	...	7.74
$7 = 2 + 5$	7	8	5.23	4.57	1.84	2.36	2.85	5.40	...	8.39
$8 = 3 + 5$	8	9	0.40	0.43	2.19	2.80	3.38	6.35	...	10.33

TABLE 3.2: *Characteristics for selected* improved *ABR correctors.*

$s = q + r$	stage order	order	β_{re}	β^*_{im}	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
$5 = 2 + 3$	6	7	2.60	3.27	2.47	3.17	3.66	6.45	...	7.85
$6 = 2 + 4$	7	8	3.84	5.85	2.05	2.63	3.20	6.04	...	8.73
$7 = 2 + 5$	8	9	5.24	8.08	1.85	2.38	2.89	5.66	...	9.55
$8 = 3 + 5$	9	10	0.97	1.39	2.20	2.82	3.42	6.56	...	11.34

order to increase the stage order by 1 we have to impose an additional condition on each row of the parameter matrices. For the r implicit and q explicit stages this could be done by filling the s^{th} column in $\underline{B}$ and the $(s-1)^{\text{th}}$ column in $\overline{A}$, respectively. The drawback of this approach is that it leads to large elements in $\overline{A}$ (for instance, if $q = 2$, $s = 6$ and $A = (a_{ij})$, then $a_{26} \approx 105$). However, it turns out that this problem does not arise in the first row. Therefore we only use this strategy for the first stage. The order of the remaining $q-1$ explicit stages will be raised by 1 by using the first column of $\overline{C}$. Remark that, strictly spoken, the last $q-1$ explicit stages become implicit in this way. In the next paragraph we will see how to handle this aspect. This approach does not lead to large coefficients and, provided that some constraints are put on the size of q and r, the sequential costs of the resulting scheme will be the same as for the ABR methods. Since these methods are much alike ABR and have a higher stage order, we will refer to them as *improved* ABR.

As a consequence of the higher order of *improved* ABR, we expect the convergence characteristics to improve. Furthermore, the stability regions should become larger than those of ABR, since *improved* ABR is 'somewhat more implicit' by the $q-1$ additional elements in $\overline{C}_1$. Comparing the Tables 3.1 and 3.2 confirm these expectations.

If we add the Radau IIA step point formula, the step point order equals $\min\{2s - 1, p+1\} = s+2$, provided that $s \geq 3$. For ABR this step point formula can be applied without any additional work, since the s^{th} stage already coincides with the last stage of the Radau IIA method. For *improved* ABR this is no longer true, since the last row of $\underline{B}$ contains a non-zero element. However, this element turns out to be very small

(for example, if $s = 7$ and $B = (b_{ij})$, then $b_{77} \approx -1.3 \cdot 10^{-12}$). Therefore in practical applications, where $s \geq 3$, we observe step point order s+2 without a step point formula (see §3.6).

3.5 The computation scheme

In this section we show how *improved* ABR methods can be implemented on r processors without any additional costs compared to ABR. The idea is to take advantage from the observation that the number of implicit stages has to exceed the number of explicit stages in order to get reasonably large stability regions.

If the number of fixed-point iterations is again denoted by m, the economized version of the ABR algorithm requires m sequential f-evaluations for the r implicit stages, plus 1 sequential f-evaluation for the q explicit stages per step. Since $r > q$, $r - q$ processors are idle during the evaluation of the explicit stages. In *improved* ABR we use these $r - q$ processors to improve the last $q - 1$ explicit stages. To see in more detail how this can be done we present the computation scheme in Table 3.3. The scheme shows which computations have to be done, categorized by matrix-vector computations (column 1) and f-evaluations (column 2). The third column denotes the number of processors that are involved in the corresponding f-evaluation.

The symbols have the following meaning:

- $\overline{A}_0$ and $\overline{B}_0$ are $q \times s$ matrices defining a slightly modified AB predictor for the first q stages: the last $q-1$ rows are the same as in AB, but the first stage is given order $s+1$ by filling the $(s-1)^{\text{th}}$ element in the first row of $\overline{A}_0$ by order conditions as well.
- $\underline{A}_0$ and $\underline{B}_0$ (both $r \times s$ matrices) are the lower parts of the AB predictor.
- $\overline{A}$, $\overline{B}$ ($q \times s$ matrices) and $\overline{C}_1$ (a $q \times q$ matrix) define a correction formula of order $s+1$ for the first q stages: $\overline{A} = \overline{A}_0$, $\overline{B}$ and the last $q-1$ components of the first column of $\overline{C}_1$ are determined by order conditions. The remaining components in $\overline{C}_1$ are 0.
- $\underline{A}$, $\underline{B}$ ($r \times s$ matrices), $\underline{C}_1$ and $\underline{C}_2$ (an $r \times q$ and $r \times r$ matrix, respectively) correspond to a correction formula of order $s+1$ by defining the last column of $\underline{A}$ and $\underline{B}$ and the whole $\underline{C}_i$ ($i \in \{1, 2\}$) by order conditions. The remaining parts of $\underline{A}$ and $\underline{B}$ are 0.
- $\overline{Y}_i^{(1)}$ and $\underline{Y}_i^{(0)}$ denote the i^{th} components of $\overline{Y}^{(1)}$ and $\underline{Y}^{(0)}$ respectively.

During the evaluation of the q components of $\overline{Y}^{(0)}$ we already evaluate the first $r - q$ components of the prediction $\underline{Y}^{(0)}$. Then we improve the last $q-1$ components of $\overline{Y}^{(0)}$ by means of the first column of $\overline{C}_1$, which results in $\overline{Y}^{(1)}$. Next we evaluate the remaining part of $\underline{Y}^{(0)}$. Now $r - q$ processors are available for the evaluation of $\overline{Y}^{(1)}$. Since we only

TABLE 3.3: *The computation scheme.*

matrix-vector computation			f-evaluations	# proc.
$\overline{Y}^{(0)}$	=	$(\overline{A}_0 \otimes I_{dd})Y_{n-1} + h(\overline{B}_0 \otimes I_{dd})F^*_{n-1}$		
$\underline{Y}^{(0)}$	=	$(\underline{A}_0 \otimes I_{dd})Y_{n-1} + h(\underline{B}_0 \otimes I_{dd})F^*_{n-1}$	$F(\overline{Y}^{(0)})$	q
			$F\begin{pmatrix} \underline{Y}^{(0)}_1 \\ \vdots \\ \underline{Y}^{(0)}_{r-q} \end{pmatrix}$	$r-q$
$\overline{Y}^{(1)}$	=	$(\overline{A} \otimes I_{dd})Y_{n-1} + h(\overline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\overline{C}_1 \otimes I_{dd})F(\overline{Y}^{(0)})$		
			$F\begin{pmatrix} \overline{Y}^{(1)}_{2q-r+1} \\ \vdots \\ \overline{Y}^{(1)}_{q} \end{pmatrix}$	$r-q$
			$F\begin{pmatrix} \underline{Y}^{(0)}_{r-q+1} \\ \vdots \\ \underline{Y}^{(0)}_{r} \end{pmatrix}$	q
$\underline{Y}^{(1)}$	=	$(\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(0)})$	$F(\underline{Y}^{(1)})$	r
$\underline{Y}^{(2)}$	=	$(\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(1)})$	$F(\underline{Y}^{(2)})$	r
$\vdots$			$\vdots$	$\vdots$
$\underline{Y}^{(m-1)}$	=	$(\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(m-2)})$	$F(\underline{Y}^{(m-1)})$	r
$\underline{Y}^{(m)}$	=	$(\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(m-1)})$		
Y_n	=	$\begin{pmatrix} \overline{Y}^{(1)} \\ \underline{Y}^{(m)} \end{pmatrix}$		
F^*_n	=	$\begin{pmatrix} F(\overline{Y}^{(1)}) \\ F(\underline{Y}^{(m-1)}) \end{pmatrix}$		
Total number of f-evaluations on r processors				$m+1$

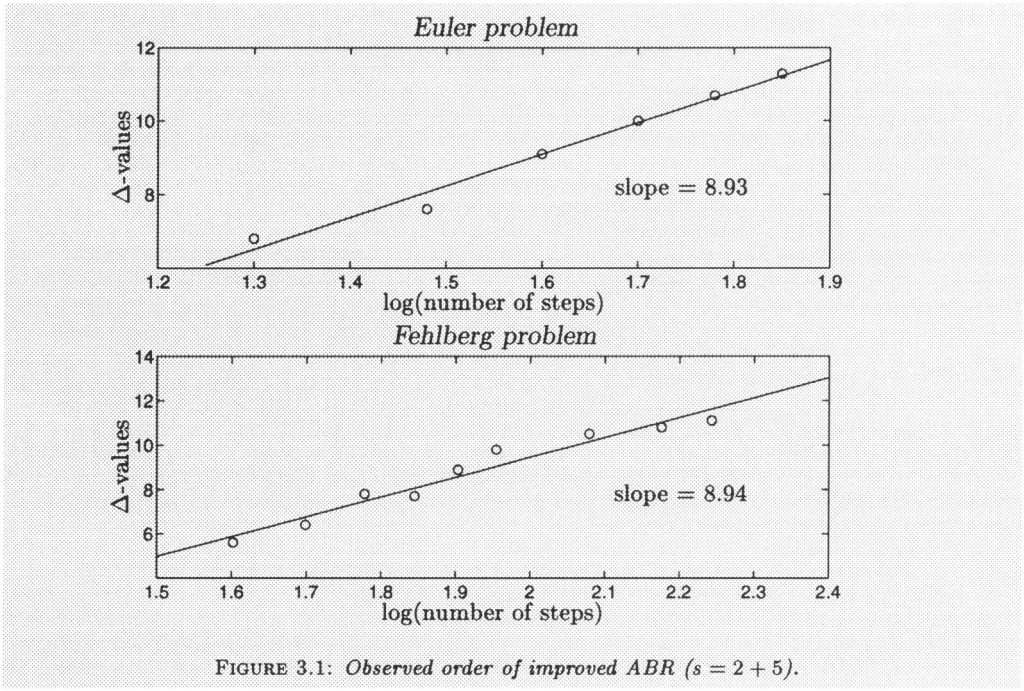

FIGURE 3.1: *Observed order of improved ABR ($s = 2 + 5$).*

benefit from the improvements in $\overline{Y}^{(1)}$ if we are able to evaluate all $q-1$ improved stages in $\overline{Y}^{(1)}$, we need to put a constraint on the size of q and r: $q - 1 \leq r - q$. Remembering the first restriction for q and r (that is, $q < r$), we conclude that for *improved* ABR q and r have to satisfy

$$q \leq \min\{r - 1, \tfrac{1}{2}(r + 1)\}. \tag{3.7}$$

Note that (3.7) holds for all the correctors in Table 3.2. The rest of the scheme is analogous to the ABR case.

From the scheme it can be seen that the first q stages are solved in PEC-mode. Numerical experiments show that just one single correction is indeed enough to solve these implicit equations.

3.6 Numerical experiments

The numerical experiments were performed using 15-digits arithmetic. The accuracies obtained are given by the number of correct digits Δ, defined by writing the maximum norm of the absolute error at the endpoint in the form $10^{-\Delta}$.

TABLE 3.4: *Comparison of f-evaluations of* improved *ABR with ABR and DOPRI8 for Euler problem.*

Δ-values	6	7	8	9	10	11
DOPRI8	415	576	728	898	1133	1422
ABR(s=2+5)	160	192	223	293	379	506
Improved ABR(s=2+5)	117	169	221	273	325	377

TABLE 3.5: *Comparison of f-evaluations of* improved *ABR with ABR and DOPRI8 for Fehlberg problem.*

Δ-values	6	7	8	9	10	11
DOPRI8	759	963	1227	1574	1990	2503
ABR(s=2+5)	335	430	532	689	846	1067
Improved ABR(s=2+5)	256	361	466	571	677	782

We took for $s = 7$ and $r = 5$, the 5-processor methods ABR (of order 8) and *improved* ABR (of order 9). We equipped both methods with the same dynamic iteration strategy (with a slightly more stringent stopping criterion) as in Chapter 2.

Two well-known test problems were taken from [HNW93], namely the the Euler problem and the Fehlberg problem which are described in (2.30)–(2.31). First we investigate to what extent the omission of the step point formula and the PEC-mode for solving the first q stages affect the observed order of *improved* ABR. Therefore we plot the Δ-values against the ^{10}log(number of steps). These points should lie on a straight line whose slope equals the step point order. Figure 3.1 shows that the expected value 9 is fairly well approximated.

Next we compare the performance of *improved* ABR with that of ABR. For completeness, we also listed the performance of the DOPRI8 code with automatic stepsize control by Hairer, Nørsett & Wanner [HNW93]. Table 3.4 and 3.5 show that *improved* ABR works about 20 % more efficiently than ABR, while the averaged speed-up factor of *improved* ABR compared to DOPRI8 (to be considered as one of the best sequential codes) is about 3.1.

3.7 Concluding remarks

The attempt to improve the parallel Adams–Bashforth–Radau (ABR) methods proposed in Chapter 2 has resulted in a more efficient code. More particularly, on 5 processors, the speed-up of the improved version compared to the fully automatic code DOPRI8 is about 3.1. This speed-up could be further improved by including a stepsize strategy.

Chapter 4

Triangularly implicit iteration methods

Abstract It often happens that iteration processes used for solving the implicit relations arising in ODE-IVP methods only start to converge rapidly after a certain number of iterations. Fast convergence right from the beginning is particularly important if we want to use so-called step-parallel iteration in which the iteration method is concurrently applied at a number of step points. In this chapter, we construct highly parallel iteration methods that do converge fast from the first iteration on. Our starting point is the PDIRK method (Parallel, Diagonally-implicit, Iterated Runge–Kutta method), designed for solving implicit Runge–Kutta equations on parallel computers. The PDIRK method may be considered as a Newton-type iteration in which the Newton Jacobian is 'simplified' to block-diagonal form. However, when applied in a step-parallel mode, it turns out that its relatively slow convergence, or even divergent behavior, reduces the effectiveness of the step-parallel scheme. By replacing the block-diagonal Newton Jacobian approximation in PDIRK by a block-triangular approximation, we do achieve convergence right from the beginning at a modest increase of the computational costs. Our convergence analysis of the block-triangular approach will be given for the wide class of general linear methods, but the derivation of iteration schemes is limited to Runge–Kutta based methods. A number of experiments show that the new parallel, triangularly implicit, iterated Runge–Kutta method (PTIRK method) is a considerable improvement over the PDIRK method.

4.1 Introduction

Suppose that we integrate the IVP

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}}$$

by an implicit step-by-step method. For the class of general linear methods (cf. [But87, p. 367]), this requires in each step the solution of a nonlinear system of the form

$$R(Y) = 0, \quad R(Y) := Y - h(A \otimes I)F(Y) - W, \tag{4.1}$$

where A denotes a nonsingular $s \times s$ matrix, W is an sd-dimensional vector containing information computed in preceding integration steps, I is the $d \times d$ identity matrix, h is the stepsize $t_n - t_{n-1}$, and $\otimes$ denotes the Kronecker product. The s components Y_i of the sd-dimensional solution vector Y represent s numerical approximations to the s exact solution vectors $y(t_{n-1} + c_i h)$; here, the c_i denote the abscissas. Furthermore, for any vector $V = (V_i)$, $F(V)$ contains the derivative values $f(V_i)$. It is assumed that the c_i are

distinct. In the following, we shall use the notation I for any identity matrix. However, its order will always be clear from the context.

The solution Y of (4.1) will be called the stage vector and s will be referred to as the number of stages. The most well-known examples of step-by-step methods that leads to implicit relations of the form (4.1), are provided by the class of implicit Runge–Kutta (RK) methods. In that case, s equals the number of *implicit* stages of the RK method. (Note that for RK methods having *explicit* stages, s is less than the *total* number of stages of the RK method, e.g. this happens for Lobatto methods.)

We want to solve the system (4.1), to be referred to as the corrector, by a parallel iteration process. Our starting point is the PDIRK method (Parallel, Diagonally-implicit Iterated Runge–Kutta method) developed in [HS91]. The PDIRK method may be considered as a Newton-type iteration in which the Newton Jacobian is 'simplified' to block-diagonal form, so that we have parallelism across the stages. Earlier comparisons on a four-processor ALLIANT FX/4 of the codes LSODE and RADAU5 (considered as belonging to the best sequential codes for stiff problems) with the PDIRK-based code PSODE (Parallel Software for ODEs) of Sommeijer [Som93], showed that in general PSODE is the most efficient one. In the average, it produces the same accuracy in half the CPU time as required by LSODE and RADAU5. In [HSV94] the PDIRK method was applied in a step-parallel setting, where the iteration procedure is concurrently applied at a number of step points, that is, iteration at the point t_{n+1} is already started without waiting until the iterates $Y^{(j)}$ at t_n have converged. (For details and analysis of various step-parallel methods we refer to e.g. [Bel87, BJZ90, Bur93a, Bur93b, Cha93, GX93, HSV94, HSV95, ML67].) This approach requires that the predictor formula needed to start iteration at t_{n+1} is based on a sufficiently 'safe' iterate $Y^{(j)}$. In order to have an efficient step-parallel iteration process, the value of j for which $Y^{(j)}$ is sufficiently 'safe' should be small, that is, substantially smaller than the order of the method (4.1). Thus, in the step-parallel approach it is particularly important that we are near convergence right from the beginning. Although the PDIRK iteration method is quite efficient when iterating until convergence, it does have the drawback of a rather slow initial convergence, and hence it is less suitable for combining it with a step-parallel approach.

The aim of the this chapter is to improve the initial convergence of the PDIRK method by replacing the Newton Jacobian $I - A \otimes hJ$ with a matrix $I - B \otimes hJ$, where B is *block-triangular* instead of block-diagonal as is in the PDIRK method. (Here, J is an approximation to the Jacobian of the righthand side function f at t_{n-1}.) The intrinsic parallelism of these 'triangularly' iterated methods is hardly less than in the PDIRK methods. The approach for determining the triangular matrix B is the same as for the PDIRK methods and is based on minimizing the spectral radius of the amplification matrix for the stiff iteration errors. However, instead of a numerical search as used for PDIRK [HS91], B can now be computed by means of the Crout decomposition of A. Although the asymptotic speed of convergence of PDIRK and the new method are often comparable, it happens that the departure from normality of the amplification matrix is considerably less than for the new methods. Hence, we may expect faster initial

convergence.

We tested the triangular iteration strategy on a few nonlinear problems from the literature. These experiments do show a considerable improvement of the initial speed of convergence for the new methods. Furthermore, we applied both methods to a rather difficult problem from circuit analysis and estimated CPU times on a four-processor Cray C98/4256 showing that the CPU time increases by less than 10%. A comparison with the RADAU5 code reveals that for this problem the new method is at least twice as fast.

4.2 The iteration scheme

Our starting point for solving the corrector equation (4.1) is the simplified (or modified) Newton iteration scheme

$$\begin{aligned} (I - A \otimes hJ)\Delta Y^{(j+1)} &= -R(Y^{(j)}), \\ Y^{(j+1)} &= Y^{(j)} + \Delta Y^{(j+1)}, \quad j = 0, 1, \ldots, \end{aligned} \tag{4.2}$$

where J is an approximation to the Jacobian of the righthand side function f at t_{n-1}, and $Y^{(0)}$ is the initial iterate to be provided by some predictor formula. Each iteration with (4.2) requires the solution of an sd-dimensional linear system for the Newton correction $\Delta Y^{(j+1)}$. In actual computation, the costs for solving this system can be reduced by first performing a similarity transformation of the iterates (cf. Butcher [But76]) $Y^{(j)} = (Q \otimes I)X^{(j)}$, where Q is a nonsingular matrix. Q should be such that the system

$$\begin{aligned} (I - Q^{-1}AQ \otimes hJ)\Delta X^{(j+1)} &= -(Q^{-1} \otimes I)R(Y^{(j)}), \\ Y^{(j+1)} &= Y^{(j)} + (Q \otimes I)\Delta X^{(j+1)}, \quad j = 0, 1, \ldots, \end{aligned} \tag{4.3}$$

is easier to solve than (4.2).

For example, if A has positive eigenvalues, then the Schur decomposition of A has the form $A = QTQ^{-1}$, where Q is orthogonal and T is lower triangular with the eigenvalues of A on its diagonal. Hence, the linear system (4.3) is 'triangularly implicit' and consists of s subsystems of dimension d that can be solved sequentially. On sequential computers, this is most effective if A has a one-point spectrum, so that only one LU-decomposition is required (see e.g. Burrage [Bur78]). On parallel computers, the condition on the spectrum of A can be relaxed to requiring that A is non-defective and has arbitrary positive eigenvalues. Since the s LU-decompositions can be computed in parallel, only one decomposition per processor is required. Similarly, in each iteration, the s forward-backward substitutions for the diagonal blocks and the s components of $R(Y^{(j)})$ can also be computed in parallel. RK methods whose RK matrices have *positive* eigenvalues can be found in Orel [Ore93].

Unfortunately, the most powerful implicit methods (with respect to order of accuracy and stability) have matrices A with *complex* eigenvalues. One option to deal with the complex eigenvalue case is to decompose A into a real block-triangular matrix of which the diagonal blocks are either diagonal or 2×2 matrices. This leads to an sd-dimensional

system that can be split into a sequence of subsystems either of dimension d or of dimension $2d$. (This approach was followed in the implementation of the RADAU5 code of Hairer and Wanner [HW91].) A block-diagonal structure of $Q^{-1}AQ$ implies that (4.3) is suitable for implementation on a parallel system.

Another option for reducing computational costs, which will be the subject of this chapter, replaces the matrix A in (4.2) by a 'more convenient' matrix B. Here, we consider the case where B is lower triangular, i.e. $B = L + D$, where L is strictly lower triangular and D is diagonal with positive diagonal entries d_{ii}. This leads to the iteration scheme

$$\begin{aligned} (I - D \otimes hJ)\Delta Y^{(j+1)} &= (L \otimes hJ)\Delta Y^{(j+1)} - R(Y^{(j)}), \\ Y^{(j+1)} &= Y^{(j)} + \Delta Y^{(j+1)}, \quad j = 0, 1, \ldots . \end{aligned} \tag{4.4}$$

In the case where L vanishes and (4.1) represents a Runge–Kutta (RK) method, the resulting iteration scheme is the PDIRK method mentioned in §4.1. The method (4.4) requires LU-decompositions of the $d \times d$ matrices $I - hd_{ii}J$, $i = 1, \ldots, s$, and, in each iteration, the evaluation of the residue $R(Y^{(j)})$, s forward-backward substitutions, and the matrix-vector multiplication $(L \otimes hJ)\Delta Y^{(j+1)}$. By expressing this multiplication in terms of F, the scheme (4.4) can be replaced by

$$(I - D \otimes hJ)\Delta Y^{(j+1)} = h(L \otimes I)(F(Y^{(j+1)}) - F(Y^{(j)})) - R(Y^{(j)}). \tag{4.5}$$

This version may yield better convergence if the righthand side Jacobian is a less accurate approximation to the true Jacobian. Just like the scheme (4.3), the LU-decompositions and the components of the residue $R(Y^{(j)})$ occurring in (4.4) and (4.5) can be evaluated in parallel. The schemes (4.4) and (4.5) will be called *parallel, triangularly implicit, iterated* methods.

In the case where (4.1) is an RK method, we shall refer to such methods as a PTIRK method and to distinguish them, we shall speak of the *LJ* and *LF version.* In the case of (4.4), a further degree of parallelism is obtained by using the Butcher similarity transformation. This enables us to eliminate the triangularly implicit term $(L \otimes hJ)\Delta Y^{(j+1)}$ and leads to

$$\begin{aligned} (I - D \otimes hJ)\Delta X^{(j+1)} &= -(Q^{-1} \otimes I)R(Y^{(j)}), \\ Y^{(j+1)} &= Y^{(j)} + (Q \otimes I)\Delta X^{(j+1)}, \quad BQ = QD\,. \end{aligned} \tag{4.6}$$

In addition to the parallelism already present in (4.4) and (4.5), the scheme (4.6) also allows that in each iteration the s forward-backward substitutions can be done in parallel. Since the schemes (4.4) and (4.6) are algebraically identical, we shall call (4.6) the *transformed LJ version.*

Finally, we compare the computational costs of the various iteration schemes. These costs consist of two contributions, respectively due to Jacobian updates and due to the successive iterations. In all schemes, the number of flops per step originating from the Jacobian updates is given by

$$C_1 = \tfrac{sd^2}{\nu}(\tfrac{1}{s}C_J + \tfrac{2}{3}d + 1),$$

TABLE 4.1: *Computational costs due to m iterations.*

Method	on one processor	on s processors
(4.4) with $L = 0$	$msd(C_f + 2d + 2s)$	$md(C_f + 2d + 2s)$
(4.4) with $L \neq 0$: LJ version	$msd(C_f + 4d + 3s)$	$md(C_f + 4ds + s^2)$
(4.5) with $L \neq 0$: LF version	$msd(C_f + 2d + 3s)$	$md(sC_f + 2ds + s^2)$
(4.6) transformed LJ version	$msd(C_f + 2d + 4s)$	$md(C_f + 2d + 4s)$

where ν denotes the averaged number of steps during which the Jacobian and the LU-decomposition is kept constant, and C_J denotes the average numbers of flops for computing one entry of J. The contribution C_1 is perfectly parallelizable and can be reduced effectively by a factor s on s processors. The contribution due to m (say) iterations are summarized in Table 4.1. In this table, C_f denotes the average numbers of flops for computing one component of f. Evidently, on a parallel computer, the methods (4.4) with $L = 0$ and (4.6) are the less expensive ones.

4.3 Convergence of the iteration process

In order to analyze convergence, we define the iteration error $\epsilon^{(j)} = Y^{(j)} - Y$, and we write the LJ and LF versions (4.4) and (4.5) in the respective forms

$$\begin{aligned}(I - B \otimes hJ)(\epsilon^{(j+1)} - \epsilon^{(j)}) &= -\epsilon^{(j)} + h(A \otimes I)(F(Y + \epsilon^{(j)}) - F(Y)), \qquad (4.7)\\ (I - D \otimes hJ)(\epsilon^{(j+1)} - \epsilon^{(j)}) &= -\epsilon^{(j)} + h((A - L) \otimes I)(F(Y + \epsilon^{(j)}) - F(Y))\\ &\quad + h(L \otimes I)(F(Y + \epsilon^{(j+1)}) - F(Y)).\end{aligned}$$

The components of $F(Y + \epsilon) - F(Y)$ can be expanded according to $J_i\epsilon_i + \mathcal{O}(\epsilon_i^2)$, where J_i is the Jacobian matrix of the righthand side function at Y_i. Assuming that J is nonsingular, we may define the block-diagonal matrix ΔJ of which the diagonal blocks are given by $J^{-1}\Delta J_i = J^{-1}(J_i - J)$ to obtain

$$F(Y + \epsilon^{(j)}) - F(Y) = (I \otimes J)\epsilon^{(j)} + (I \otimes J)\Delta J\epsilon^{(j)} + \mathcal{O}((\epsilon^{(j)})^2).$$

Ignoring the second-order terms (first-order convergence analysis), the error recursions for the LJ and LF versions can be represented in the forms

$$\epsilon^{(j+1)} = M(I + P\Delta J)\epsilon^{(j)}, \qquad (4.8)$$

$$\epsilon^{(j+1)} = (I - N\Delta J)^{-1}M(I + Q\Delta J)\epsilon^{(j)}, \qquad (4.9)$$

where

$$\begin{aligned}M &:= (I - B \otimes hJ)^{-1}((A - B) \otimes hJ),\\ N &:= (I - B \otimes hJ)^{-1}(L \otimes hJ),\\ P &:= (A - B)^{-1}A \otimes I, \quad Q := (A - B)^{-1}(A - L) \otimes I\,.\end{aligned}$$

If we ignore ΔJ (linear convergence analysis), then the error recursions of both versions are characterized by the matrix M. However, if ΔJ cannot be neglected, then the error recursions may behave quite differently. For example, as $h \to 0$, then we have

$$\epsilon^{(j+1)} \approx h((A-B)\otimes J + (A\otimes J)\Delta J)\epsilon^{(j)}, \tag{4.10}$$

$$\epsilon^{(j+1)} \approx h((A-B)\otimes J + ((A-L)\otimes J)\Delta J)\epsilon^{(j)}. \tag{4.11}$$

Since the strictly lower triangular blocks of the amplification matrices in (4.10) and (4.11) differ by the matrices $hL_{ij}\Delta J_j$, the convergence behavior may differ considerably and is highly problem dependent. In the remainder of this chapter, we shall focus on the matrix M.

4.3.1 Rate of convergence

In order to select a suitable matrix B, we consider the convergence of the individual error components corresponding to the eigenvalues λ of J. From (4.7) it follows that these error components are amplified by the matrix Z defined by

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

Z will be called the *amplification matrix associated with* M. A measure for the rate of convergence of the individual error components is defined by the (averaged) amplification factors

$$\rho_j(z) := \sqrt[j]{\|Z(z)^j\|_\infty}. \tag{4.12}$$

where $\|\cdot\|_\infty$ denotes the maximum norm. Note that $\rho_\infty(z) = \rho(Z(z))$, $\rho(\cdot)$ being the spectral radius function. For the test equation $y' = \lambda y$, the value of $\rho_j(z)$ may be interpreted as the averaged factor by which the iteration error corresponding to $z = h\lambda$ is reduced in each iteration, until the corrector solution is reached. For more general problems, we have to deal with $\rho_j(z)$ where z runs through the spectrum of hJ.

The amplification factor at $z = \infty$ will be called the *stiff* amplification factor. In the neighborhood of the origin we may write

$$\rho_j(z) = |z|\sqrt[j]{\|(A-B)^j\|_\infty} + \mathcal{O}(z^2) =: \widetilde{\rho}_j(z) + \mathcal{O}(z^2) \text{ as } z \to 0. \tag{4.13}$$

The quantity $\widetilde{\rho}_j(z)$ will be called the *non-stiff* amplification factor. Furthermore, we denote the maximal amplification factor in the left-hand plane by ρ_j^*. Of course, ρ_j^* refers to the worst case situation, but it serves as an indicator for the robustness of the method.

4.3.2 Iteration strategies

In this section, we discuss the choice of the free matrix $B = L + D$ in the iteration schemes (4.4) and (4.5). We first briefly review the *diagonal* iteration strategy of [HS91], i.e. $L = 0$, and then we focus on the *triangular* iteration strategy where L is allowed to be an arbitrary strictly (lower) *triangular* matrix. A nonvanishing matrix L enables us to reduce the norm of Z^j considerably.

The reason for restricting B to the class of triangular matrices is that we have direct control over the eigenvalues of B. As a consequence, suitable matrices B can be constructed without performing a many-parameter search as was carried out in [HS91]. Since our main source of correctors is the class of RK methods which usually possess a dominant *lower* triangular part, B is also assumed to be *lower* triangular. (Recall that ideally B should equal A.) Both for the diagonal iteration and the triangular iteration approach, the matrices B, $Z_0 := A - B$ and $Z_\infty := I - B^{-1}A$ associated with a number of classical RK methods are specified in the appendix to this chapter.

Diagonal iteration

The diagonal iteration strategy is characterized by a diagonal matrix B with positive diagonal entries. In this strategy, it was found for a large number of classical RK correctors that small iteration error amplification factors for the stiff error components are crucial for a satisfactory overall convergence [HS91]. This is due to an order reduction effect common in stiff situations and can be explained by considering the error recursions (4.8) and (4.9). Due to the 'Jacobian-defect' matrix ΔJ, the stiff error components are not damped as strongly as the non-stiff error components. Therefore, we determined in [HS91] the diagonal matrix $B = D$ such that Z_∞ has a minimal spectral radius; that is, the *asymptotic* value $\rho_\infty(\infty) = \rho(Z_\infty)$ of the stiff amplification factor is minimized. In [HS91] this was achieved by a multi-parameter search over the diagonal entries of D. For a large number of collocation based RK correctors, it turned out that the spectral radius $\rho(Z_\infty)$ of $Z_\infty = I - B^{-1}A$ is extremely small. In fact, we conjecture that for collocation based RK correctors, there exist matrices D with positive diagonal entries for which $\rho(Z_\infty)$ actually vanishes. This suggests an alternative construction of the matrix $B = D$. Writing down the characteristic equation for Z_∞ and imposing the condition that this equation has only zero roots, we arrive at a (nonlinear) system for the entries d_{ii} of D. If this system can be solved for positive d_{ii}, $i = 1, \ldots, s$, then we have found an optimal matrix D. It has been verified for the Radau IIA correctors with $s \leq 8$ that such optimal matrices D do exist (see [Lio96]). Notice that a zero spectral radius $\rho(Z_\infty)$ implies that Z_∞^j vanishes for $j \geq s$. (This can be seen by considering the Schur decomposition $Z_\infty = QTQ^{-1}$ with Q orthogonal and T strictly lower triangular.)

If the matrices D are obtained by a numerical search as in [HS91], then they will always give rise to a small but yet nonzero $\rho(Z_\infty)$. Nevertheless, both for the *non-stiff* and the *highly stiff* error components, the generated PDIRK methods show a satisfactory convergence rate for larger values of j. On the other hand, it also turns out that for the higher-order methods, there may be regions in the z-plane where $\rho_j(z)$ exceeds one for small j, so that initially error components corresponding to points lying in such regions will diverge [HSV94, Tables 3.2b]. The reason for this behavior is the 'abnormality' of the matrix Z. In particular, for larger values of $|z|$, i.e. for the stiff error components, the matrix $Z(z)$ may differ considerably from a normal matrix. To be more precise, let the departure from normality of the matrix Z be defined by $\Delta^2(Z) := \|Z\|_F^2 - \|\zeta(Z)\|_2^2$, where $\zeta(Z)$ denotes the vector of eigenvalues of Z and $\|\cdot\|_F$ and $\|\cdot\|_2$, respectively, denote the Frobenius matrix norm and the Euclidean vector norm (see e.g. [GL89, p. 336]). By

considering plots of $\Delta^2(Z)$ as a function of $|z|$ with $\arg(z)$ constant, we found that in the left-hand half-plane $\Delta^2(Z)$ monotonically increases from 0 to values greater than 20. This situation is particularly unfortunate if we want to apply the step-parallel iteration approach mentioned in §4.1. In such an approach, it is crucial that in the whole left-hand half-plane the amplification factor is less than one right from the beginning.

Triangular iteration

In the triangular iteration strategy we choose B lower triangular with positive diagonal entries such that Z_∞ is *strictly upper triangular*. As a consequence, we have a zero stiff amplification factor for $j \geq s$. For the construction of such a matrix B we use the LU-decomposition of A. Let $A = T_L T_U$ with T_L lower triangular and T_U unit upper triangular (Crout decomposition), and define $B = T_L$. Since $Z_\infty = I - B^{-1}A$, we immediately obtain the strictly upper triangular matrix $Z_\infty = I - T_U$. The following lemma provides an explicit criterion for the positiveness of the diagonal entries of $B = T_L$.

LEMMA 4.1 *Let A, L, D, and U be $s \times s$ matrices such that $A = LDU$ with L unit lower triangular, D diagonal and U unit upper triangular, and let A_k denote the $k \times k$ principal submatrix of A. Then D has diagonal entries given by*

$$d_k = \frac{\det(A_k)}{\det(A_{k-1})}, \quad k = 1, \ldots, s, \tag{4.14}$$

where $\det(A_0) := 1$ *and* $\det(A_1) := a_{11}$.

PROOF Let A_i be decomposed according to $A_k = L_k D_k U_k$ with L_k unit lower triangular, D_k diagonal and U_k unit upper triangular. Then

$$\det(A_k) = \det(L_k)\det(D_k)\det(U_k) = \det(D_k).$$

Since the first $k-1$ pivots in the Gaussian elimination process do not depend on the entries a_{ij} with $i \geq k$ and $j \geq k$, it follows that the diagonal entries d_{ik} of D_k are defined by $d_{ik} = d_i$. Hence, $\det(D_k) = \det(D_{k-1})d_k$, which is equivalent with (4.14). □

From this lemma it follows that the diagonal entries of the matrix B defined above are given by (4.14), so that they are all positive, if all values $\det(A_k)$, $k = 1, \ldots, s$, are positive. In the following, we restrict our considerations to collocation methods with distinct abscissas c_i. Such methods are generated by matrices A of the form (see, e.g., [HW91, p. 82])

$$\begin{array}{lll} A = CVRV^{-1}, & C = \mathrm{diag}\{c\}, & R = \mathrm{diag}\{r\}, \\ c = (c_i), & r = (i^{-1}), & V = \left[\mathbb{1}\, c\, c^2 \cdots c^{s-1}\right], \end{array} \tag{4.15}$$

where $i = 1, \ldots, s$ and $\mathbb{1}$ is the vector with unit entries.

THEOREM 4.1 *If A results from a collocation method with positive, distinct abscissas ordered according to $0 < c_1 < c_2 < \cdots < c_s$, then the following results hold:*

1. *The values of* $\det(A_1)$ *and* $\det(A_s)$ *are positive for all* s.

2. *Let* V *and* R *be partitioned according to (4.16), where* V_k *and* R_k *denote the* $k \times k$ *principal submatrices of the matrices* V *and* R *defined in (4.15). Then, for* $1 < k < s$, $\det(A_k)$ *is positive if*

$$q_k := \det(V_k R_k - PSW^{-1}Q) > 0, \quad V = \begin{bmatrix} V_k & P \\ Q & W \end{bmatrix}, \quad R = \begin{bmatrix} R_k & 0 \\ 0 & S \end{bmatrix}. \tag{4.16}$$

PROOF

1. For collocation methods we have that

$$a_{11} = \int_0^{c_1} \frac{c_2 - t}{c_2 - c_1} \frac{c_3 - t}{c_3 - c_1} \cdots \frac{c_s - t}{c_s - c_1} \mathrm{d}t.$$

From the condition on the collocation points it is immediate that $\det(A_1) = a_{11} > 0$ and from (4.15) it follows that $\det(A) = \det(C)\det(VRV^{-1}) = \det(C)\det(R) > 0$.

2. By means of (4.15) it is easily verified that A_k can be presented in the form

$$A_k = C_k(V_k R_k - PSW^{-1}Q)(V_k - PW^{-1}Q)^{-1}, \tag{4.17}$$

where C_k is the $k \times k$ principal submatrix of C and V_k, R_k, P, S, W and Q are specified in (4.16). We now prove that $\det(V_k - PW^{-1}Q)$ is positive by considering the factorizations

$$\begin{bmatrix} V_k - PW^{-1}Q & 0 \\ Q & W \end{bmatrix} = \begin{bmatrix} I & -PW^{-1} \\ 0 & I \end{bmatrix} V,$$

$$\begin{bmatrix} V_k - PW^{-1}Q & 0 \\ Q & W \end{bmatrix} = \begin{bmatrix} V_k - PW^{-1}Q & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ Q & W \end{bmatrix}.$$

From these two relations it follows that $\det(V) = \det(V_k - PW^{-1}Q)\det(W)$. Since V is a VanderMonde matrix and W is a row-scaled VanderMonde matrix, we conclude that both V and W have a positive determinant. Thus, $\det(V_k - PW^{-1}Q) > 0$, and by virtue of (4.17), it follows that $\det(A_k)$ is positive whenever the quantity q_k defined in (4.16) is positive. □

Theorem 4.1 directly implies the positiveness of the diagonal entries of $B = T_L$ for all two-stage collocation methods. For higher-stage methods, it provides the relatively simple criterion $q_k > 0$, $1 < k < s$, for verifying the condition $\det(A_k) > 0$. We conjecture that the condition $\det(A_k) > 0$, $1 \leq k \leq s$, is true for all s, but so far we are not able to prove it. Instead, we verified the correctness of this conjecture for $s \leq 6$. An easy way of verifying the conditions $q_k > 0$, replaces the abscissas c_i in q_k by $c_i = p_i + p_{i-1} + \cdots + p_1$

and expresses q_k as a rational function of the s parameters p_i. For $s \leq 6$, it turns out that all coefficients in this rational expression are positive. Since the parameters p_i are all positive (because $p_i := c_i - c_{i-1}$ with $c_0 := 0$), this implies that q_k is positive.

EXAMPLE For $s = 3$, we have to prove that $q_2 > 0$. A straightforward calculation yields

$$q_2 = \frac{3p_2p_3^2 + 4p_2^2p_3 + 2p_1p_2p_3 + p_1p_2^2 + p_2^3}{6(p_1 + p_2 + p_3)^2},$$

which is obviously positive. ◇

TABLE 4.2: *Amplification factors* $\widetilde{\rho}_j$, $\rho_j(\infty)$ *and* ρ_j^*.

	Method	Gauss	Lobatto IIIA		Radau IIA		
		$s = 2$	$s = 2$	$s = 3$	$s = 2$	$s = 3$	$s = 4$
$\widetilde{\rho}_1(z)$	PDIRK	$0.79\|z\|$	$0.89\|z\|$	$0.91\|z\|$	$1.15\|z\|$	$1.15\|z\|$	$1.10\|z\|$
	PTIRK	$0.08\|z\|$	$0.08\|z\|$	$0.13\|z\|$	$0.15\|z\|$	$0.21\|z\|$	$0.25\|z\|$
$\widetilde{\rho}_2(z)$	PDIRK	$0.36\|z\|$	$0.31\|z\|$	$0.32\|z\|$	$0.52\|z\|$	$0.44\|z\|$	$0.52\|z\|$
	PTIRK	$0.08\|z\|$	$0.08\|z\|$	$0.13\|z\|$	$0.15\|z\|$	$0.20\|z\|$	$0.22\|z\|$
$\widetilde{\rho}_3(z)$	PDIRK	$0.45\|z\|$	$0.21\|z\|$	$0.30\|z\|$	$0.40\|z\|$	$0.34\|z\|$	$0.25\|z\|$
	PTIRK	$0.08\|z\|$	$0.08\|z\|$	$0.12\|z\|$	$0.15\|z\|$	$0.20\|z\|$	$0.20\|z\|$
$\rho_1(\infty)$	PDIRK	1.58	2.29	5.62	1.78	4.17	4.68
	PTIRK	0.15	0.13	0.23	0.20	0.46	0.68
$\rho_2(\infty)$	PDIRK	0	0	3.09	0	1.82	3.31
	PTIRK	0	0	0.17	0	0.26	0.47
$\rho_3(\infty)$	PDIRK	0	0	0	0	0	2.09
	PTIRK	0	0	0	0	0	0.30
ρ_1^*	PDIRK	1.58	2.29	5.62	1.78	4.17	4.68
	PTIRK	0.15	0.13	0.23	0.20	0.46	0.68
ρ_2^*	PDIRK	0.59	0.58	3.09	0.63	1.82	3.31
	PTIRK	0.14	0.14	0.33	0.18	0.40	0.58
ρ_3^*	PDIRK	0.45	0.39	1.35	0.47	1.02	2.09
	PTIRK	0.14	0.14	0.32	0.18	0.39	0.55
ρ_∞^*	PDIRK	0.25	0.17	0.45	0.26	0.40	0.52
	PTIRK	0.14	0.14	0.30	0.18	0.37	0.50

Summarizing we conclude that, unlike the diagonal approach, the triangular approach provides an extremely simple construction of the matrix B and an explicit criterion for checking the positiveness of its diagonal entries. Moreover, it turns out that for larger values of $|z|$ the departure from normality $\Delta^2(Z) := \|Z\|_F^2 - \|\zeta(Z)\|_2^2$ is considerably reduced. This can be explained by the fact that the magnitude of the entries of the matrix Z (and hence $\|Z\|_F^2$) can be made much smaller by a triangular matrix B than by a diagonal matrix B. (Recall that Z contains the factor $A - B$.) Plots show that $\Delta^2(Z)$ monotonically increases from zero at the origin to values less than than 0.4 at infinity, resulting in amplification factors that are less than one in the whole left-hand half-plane for all j. This will be quantified in the following subsection.

TABLE 4.3: *LSV predictor: HIRES problem of Schäfer.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	15	*	*	*	4.3	$\cdots$	6.5
PTIRK(LJ)	15	3.4	3.5	3.8	4.2	$\cdots$	6.3
PTIRK(LF)	15	3.1	4.0	3.9	4.1	$\cdots$	5.6
PDIRK	7.5	*	*	*	5.4	$\cdots$	7.7
PTIRK(LJ)	7.5	4.0	4.2	4.7	5.1	$\cdots$	8.3
PTIRK(LF)	7.5	3.3	4.4	4.7	5.3	$\cdots$	7.0

TABLE 4.4: *EPL predictor: HIRES problem of Schäfer.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	15	*	*	*	*	$\cdots$	6.4
PTIRK(LJ)	15	*	3.0	4.8	5.1	$\cdots$	7.3
PDIRK	7.5	*	*	*	4.1	$\cdots$	8.8
PTIRK(LJ)	7.5	*	2.5	6.1	6.6	$\cdots$	9.0

4.3.3 Comparison of amplification factors

For a number of well-known RK correctors, we compare the amplification factors defined by (4.12) associated with the diagonal approach (PDIRK method) and the triangular approach (PTIRK method). For $j = 1, 2, 3$, Table 4.2 presents the *non-stiff* amplification factor $\widetilde{\rho}_j(z)$ as defined in (4.13), the *stiff* amplification factor $\rho_j(\infty)$, and the maximal amplification factor ρ_j^*. These figures indicate that initially, the PTIRK strategy converges considerably faster than the PDIRK strategy. Hence, it should be a sound starting point for step-parallel applications. This will be subject of future research.

4.4 Numerical illustration

In this section, we compare the diagonally-implicit iteration (PDIRK) strategy with the triangularly implicit iteration (PTIRK) strategy. In all experiments, we used the four-stage Radau IIA corrector with constant stepsizes and a Jacobian update in each step. If necessary, the initial condition in the problems below is adapted such that the integration starts outside the transient phase enabling us to use constant steps. Two predictors were tested, the simple last step value (LSV) predictor $Y^{(0)} = (e_s^{\mathrm{T}} \otimes I)Y$ and the extrapolation (EPL) predictor $Y^{(0)} = (E \otimes I)Y$. Here, Y denotes the stage vector from the preceding step, e_s is the s^{th} unit vector, and E is the extrapolation matrix. In the following two subsections we compare the accuracy and the CPU time on a four-processor Cray C98/4256 of the PDIRK and PTIRK methods.

TABLE 4.5: *LSV predictor: Chemical reaction problem of Gear.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	50	1.4	2.2	2.6	2.9	$\cdots$	5.2
PTIRK(LJ)	50	2.3	2.7	3.5	4.3	$\cdots$	7.7
PTIRK(LF)	50	1.8	2.9	3.9	3.0	$\cdots$	3.3
PDIRK	25	1.8	2.9	3.4	3.6	$\cdots$	7.3
PTIRK(LJ)	25	2.3	3.6	4.2	5.3	$\cdots$	9.8
PTIRK(LF)	25	2.1	4.3	4.4	4.6	$\cdots$	6.4

TABLE 4.6: *EPL predictor: Chemical reaction problem of Gear.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	25	2.4	2.8	3.2	3.6	$\cdots$	7.4
PTIRK(LJ)	25	2.9	3.7	4.3	5.6	$\cdots$	9.8

4.4.1 Accuracy tests

In the tables of results, the LF and LJ versions (4.4) and (4.5) of the PTIRK method are indicated by PTIRK(LJ) and PTIRK(LF). For a given number of iterations m, the tables of results below present the minimal number of correct digits cd of the components of the numerical solution at the end point $t = t_{\text{end}}$ of the integration interval, that is, at the end point the absolute errors are written as 10^{-cd}. Our first example (Tables 4.3 and 4.4) is provided by a problem of Schäfer, called the HIRES problem in [HW91, p. 157]. It was proposed in Gottwald [Got77] as a test problem and consists of eight mildly stiff equations on the interval $5 \leq t \leq 305$. (It is included into the CWI test set [LSV96].) The second test problem (Tables 4.5 and 4.6) is a set of three chemical reaction equations originating from Gear [Gea69] on the interval [1,51] and is included in the test set of Enright et al. [EHL75]. The ATMOS20 problem is our third test problem (Tables 4.7 and 4.8). It is a system of 20 stiff nonlinear ODEs originating from an air pollution model used by Verwer [Ver94] and included in the CWI test set [LSV96]. We solved this system in the integration interval [5,60]. The tables of results clearly show for both predictors the superiority of the PTIRK strategy in the first few iterations. For large numbers of iterations, PDIRK and PTIRK(LJ) are better than PTIRK(LF).

4.4.2 Cray C98/4256 tests

We applied the PDIRK and PTIRK(LJ) methods on a four-processor Cray C98/4256 to the Ring modulator of Horneber. This problem consists of 15 highly stiff differential equations on the interval [0,0.001]. (For details, see the CWI test set [LSV96].) PDIRK and PTIRK(LJ) were applied with the EPL predictor and stepsize $h = 1.25 \cdot 10^{-7}$. We compiled the codes with `cf77` using the flags `-dp`, `-Zp`, `-Wu-p`, and `-Wd-dj`. The

TABLE 4.7: *LSV predictor: ATMOS20 problem of Verwer.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	11	2.7	2.1	2.8	5.4	$\cdots$	9.8
PTIRK(LJ)	11	3.3	5.0	6.1	6.7	$\cdots$	11.0
PTIRK(LF)	11	3.4	4.9	7.0	6.8	$\cdots$	8.7
PDIRK	5.5	1.3	*	*	6.5	$\cdots$	11.2
PTIRK(LJ)	5.5	3.7	5.6	7.0	7.7	$\cdots$	12.2
PTIRK(LF)	5.5	3.7	5.5	7.6	8.3	$\cdots$	11.5
PDIRK	2.25	*	*	*	7.5	$\cdots$	12.2
PTIRK(LJ)	2.25	4.0	6.2	7.8	8.6	$\cdots$	12.6
PTIRK(LF)	2.25	4.0	6.2	8.2	10.0	$\cdots$	12.1

TABLE 4.8: *EPL predictor: ATMOS20 problem of Verwer.*

Method	h	$m=1$	$m=2$	$m=3$	$m=4$	$\cdots$	$m=10$
PDIRK	11	1.8	2.6	2.1	5.4	$\cdots$	10.0
PTIRK(LJ)	11	2.0	3.7	6.3	7.0	$\cdots$	10.9
PDIRK	5.5	*	*	*	6.4	$\cdots$	11.8
PTIRK(LJ)	5.5	*	4.2	7.4	8.2	$\cdots$	12.2
PDIRK	2.25	*	*	*	8.2	$\cdots$	12.7
PTIRK(LJ)	2.25	*	*	8.6	9.4	$\cdots$	12.7

environmental variable `NCPUS` had the value 4. For the meaning of these settings, we refer to Cray Research Inc., CF77 Commands and Directives, SR-3771, 6.0 edition, 1994. Table 4.9 lists the *cd*-values and the CPU(1) and CPU(4) timings (in sec.) required on one and four processors, respectively. For the CPU(1) timings we used the internal function `ETIME` and the CPU(4)-values were obtained by using the Cray tool `ATExpert`. These figures again show that PTIRK(LJ) is much more accurate than PDIRK, while it is hardly more expensive than PDIRK (less than 10%). Since earlier experiments on a four-processor ALLIANT FX/4 indicated that the PDIRK-based code PSODE is in general twice as efficient as LSODE and RADAU5 [Som93, p. 12], we expect that a PTIRK-based code should be at least twice as efficient as LSODE and RADAU5. The present version of the PTIRK code does not yet contain a sufficiently tested stepsize and iteration-stopping strategy. Therefore it is not yet possible to compare it with codes like LSODE and RADAU5. Nevertheless, in order to have some indication how PTIRK performs in a parallel environment, we applied RADAU5 with the same integration strategy as our present PTIRK code, that is, with constant stepsizes (without step rejections) and with a Jacobian update in each step. In order to generate a range of *cd*-values, we run RADAU5 with a few different stepsizes. The *cd*-values and CPU(1) timings obtained are also listed in Table 4.9.

We conclude this chapter with the following:

TABLE 4.9: *Ring modulator of Horneber.*

Method		$M = 2$	$m = 3$	$m = 4$	...	$m = 10$
PDIRK	*cd*	*	*	4.6	...	8.5
	CPU(1)	*	*	26.2	...	51.9
	CPU(4)	*	*	8.2	...	16.7
PTIRK(LJ)	*cd*	5.7	7.7	8.3	...	8.5
	CPU(1)	20.1	23.1	28.1	...	56.6
	CPU(4)	6.1	7.0	8.8	...	18.3
RADAU5	*cd*	5.8	6.5	7.2		
	CPU(1)	14.0	17.5	21.9		
	$10^7 h$	1.25	1.0	0.8		

1. For a relatively difficult problem as provided by the Ring modulator, the PTIRK code on a Cray in four-processor mode shows a speed-up ranging from > 2.4 to > 3.1 with respect to RADAU5 on a Cray in one-processor mode.

2. It is not expected that a parallel implementation of RADAU5 is as efficient as PTIRK, because the intrinsic parallelism of PTIRK is much larger than that of RADAU5. For example, the effective LU-costs and forward-backward substitutions are four times as expensive. This is due to the fact that the eigenvalues of the Radau IIA matrix are not all real, so that the Butcher transformation used in RADAU5 decouples the $3d$-dimensional system into one real and one *complex* system of dimension d. The LU-decompositions and the forward-backward substitutions associated with these systems can be done concurrently. However, complex arithmetic is about four times as expensive as real arithmetic, which explains the factor 4 mentioned above.

Appendix

For a number of RK methods, we have computed the matrices $B = L + D$ according to the procedure outlined in §4.3.2, together with the amplification matrices Z_0 and Z_∞.

PDIRK strategy: $Z(z) = z(I - zD)^{-1}(A - D)$, $Z_0 = A - D$, $Z_\infty = I - D^{-1}A$.

Radau IIA

$s = 2$:

$$D = \begin{bmatrix} 0.2584 & 0 \\ 0 & 0.6449 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0.1582 & -0.0833 \\ 0.7500 & -0.3949 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} -0.6124 & 0.3225 \\ -1.1629 & 0.6124 \end{bmatrix}.$$

$s = 3$:

$$D = \begin{bmatrix} 0.3204 & 0 & 0 \\ 0 & 0.1400 & 0 \\ 0 & 0 & 0.3717 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} -0.1236 & -0.0655 & 0.0238 \\ 0.3944 & 0.1521 & -0.0415 \\ 0.3764 & 0.5125 & -0.2606 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0.3857 & 0.2045 & -0.0742 \\ -2.8179 & -1.0867 & 0.2968 \\ -1.0127 & -1.3789 & 0.7011 \end{bmatrix}.$$

$s = 4$:

$$D = \begin{bmatrix} 0.3205 & 0 & 0 & 0 \\ 0 & 0.0892 & 0 & 0 \\ 0 & 0 & 0.1817 & 0 \\ 0 & 0 & 0 & 0.2334 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} -0.2075 & -0.0403 & 0.0258 & -0.0099 \\ 0.2344 & 0.1177 & -0.0479 & 0.0160 \\ 0.2167 & 0.4061 & 0.0073 & -0.0242 \\ 0.2205 & 0.3882 & 0.3288 & -0.1709 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0.6474 & 0.1258 & -0.0805 & 0.0309 \\ -2.6290 & -1.3206 & 0.5368 & -0.1800 \\ -1.1923 & -2.2346 & -0.0402 & 0.1331 \\ -0.9447 & -1.6635 & -1.4092 & 0.7322 \end{bmatrix}.$$

Lobatto IIIA

$s = 2$:

$$D = \begin{bmatrix} 0.2113 & 0 \\ 0 & 0.3943 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0.1220 & -0.0417 \\ 0.6667 & -0.2277 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} -0.5774 & 0.1972 \\ -1.6906 & 0.5774 \end{bmatrix}.$$

$s = 3$:

$$D = \begin{bmatrix} 0.4802 & 0 & 0 \\ 0 & 0.1094 & 0 \\ 0 & 0 & 0.1604 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} -0.2905 & -0.0339 & 0.0103 \\ 0.4506 & 0.1175 & -0.0270 \\ 0.4167 & 0.4167 & -0.0770 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0.6049 & 0.0706 & -0.0215 \\ -4.1179 & -1.0743 & 0.2465 \\ -2.5981 & -2.5981 & 0.4804 \end{bmatrix}.$$

Gauss

$s = 2$:

$$D = \begin{bmatrix} 0.1667 & 0 \\ 0 & 0.5000 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0.0833 & -0.0387 \\ 0.5387 & -0.2500 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} -0.5000 & 0.2321 \\ -1.0774 & 0.5000 \end{bmatrix}.$$

PTIRK strategy: $Z(z) = z(I - zB)^{-1}(A - B), \quad Z_0 = A - B, \quad Z_\infty = I - B^{-1}A.$

Radau IIA

$s = 2$:

$$B = \begin{bmatrix} 0.4167 & 0 \\ 0.7500 & 0.4000 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & -0.0833 \\ 0 & -0.1500 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.2000 \\ 0 & 0 \end{bmatrix}.$$

$s = 3$:

$$B = \begin{bmatrix} 0.1968 & 0 & 0 \\ 0.3944 & 0.4234 & 0 \\ 0.3764 & 0.6378 & 0.2000 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & -0.0655 & 0.0238 \\ 0 & -0.1313 & -0.0415 \\ 0 & -0.1253 & -0.0889 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.3330 & -0.1208 \\ 0 & 0 & 0.2106 \\ 0 & 0 & 0 \end{bmatrix}.$$

$s = 4$:

$$B = \begin{bmatrix} 0.1130 & 0 & 0 & 0 \\ 0.2344 & 0.2905 & 0 & 0 \\ 0.2167 & 0.4834 & 0.3083 & 0 \\ 0.2205 & 0.4668 & 0.4414 & 0.1176 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & -0.0403 & 0.0258 & -0.0099 \\ 0 & -0.0836 & -0.0479 & 0.0160 \\ 0 & -0.0773 & -0.1192 & -0.0242 \\ 0 & -0.0786 & -0.1126 & -0.0551 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.3567 & -0.2283 & 0.0877 \\ 0 & 0 & 0.3490 & -0.1260 \\ 0 & 0 & 0 & 0.2144 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Lobatto IIIA

$s = 2$:

$$B = \begin{bmatrix} 0.3333 & 0 \\ 0.6667 & 0.2500 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & -0.0417 \\ 0 & -0.0833 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.1250 \\ 0 & 0 \end{bmatrix}.$$

$s = 3$:

$$B = \begin{bmatrix} 0.1897 & 0 & 0 \\ 0.4506 & 0.3075 & 0 \\ 0.4167 & 0.4911 & 0.1429 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & -0.0339 & 0.0103 \\ 0 & -0.0805 & -0.0270 \\ 0 & -0.0745 & -0.0595 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.1787 & -0.0543 \\ 0 & 0 & 0.1673 \\ 0 & 0 & 0 \end{bmatrix}.$$

Gauss

$s = 2$:

$$B = \begin{bmatrix} 0.2500 & 0 \\ 0.5387 & 0.3333 \end{bmatrix}, \quad Z_0 = \begin{bmatrix} 0 & 0.0387 \\ 0 & 0.0833 \end{bmatrix},$$

$$Z_\infty = \begin{bmatrix} 0 & 0.1547 \\ 0 & 0 \end{bmatrix}.$$

Chapter 5

Approximating Runge–Kutta matrices by triangular matrices

Abstract The implementation of implicit Runge–Kutta methods requires the solution of large systems of non-linear equations. Normally these equations are solved by a modified Newton process, which can be very expensive for problems of high dimension. The in Chapter 4 proposed triangularly implicit iteration methods for ODE-IVP solvers substitute the Runge–Kutta matrix A in the Newton process for a triangular matrix T that approximates A, hereby making the method suitable for parallel implementation. The matrix T is constructed according to a simple procedure, such that the stiff error components in the numerical solution are strongly damped. In this chapter we prove for a large class of Runge–Kutta methods that this procedure can be carried out and that the diagonal entries of T are positive. This means that the linear systems that are to be solved have a non-singular matrix.

5.1 Introduction and motivation

For solving the stiff initial value problem

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}},$$

one of the most powerful methods is an implicit Runge–Kutta (RK) method. In such a method we have to solve every time step a system of non-linear equations of the form

$$R(Y_n) = 0; \quad R(Y_n) := Y_n - (\mathbb{1} \otimes I)y_{n-1} - h_n(A \otimes I)F(Y_n), \tag{5.1}$$

where A denotes the $s \times s$ matrix containing the parameters of the s-stage RK method, y_{n-1} the approximation to $y(t_{n-1})$, $\mathbb{1}$ is the s-dimensional vector with unit entries, I is the $d \times d$ identity matrix, h_n is the stepsize $t_n - t_{n-1}$ and $\otimes$ denotes the Kronecker product. The s components $Y_{n,i}$ of the sd-dimensional solution vector Y_n represent s numerical approximations to the s exact solution vectors $y(\mathbb{1}t_{n-1} + c_i h_n)$; here, c denotes the abscissa vector and i ranges from 1 to s. Furthermore, for any vector $X = (X_i)$, $F(X)$ contains the derivative values $f(X_i)$. It is assumed that the components of c are distinct and positive.

Once we have solved (5.1), we obtain the step point value $y_n \approx y(t_n)$ by the formula

$$y_n = y_{n-1} + h_n(b^{\mathrm{T}} \otimes I)F(Y_n),$$

where b is a vector of dimension s containing method parameters.

To solve (5.1), in general one uses a Newton-type iteration scheme of the form

$$(I - B \otimes h_n J_n)\Delta Y_n^{(j+1)} = -R(Y_n^{(j)}); \quad Y_n^{(j+1)} = Y_n^{(j)} + \Delta Y_n^{(j+1)}, \tag{5.2}$$

where J_n is an approximation to the Jacobian of the right-hand side function f at t_{n-1}, $Y_n^{(0)}$ is the initial iterate to be provided by some predictor formula and B is an $s \times s$ matrix that defines the type of Newton iteration. To get insight in the convergence behavior of (5.2), we apply the scheme to the scalar test equation $y' = \lambda y$. Defining the iteration error $\epsilon_n^{(j)}$ by $Y_n^{(j)} - Y_n$, we see from (5.1) and (5.2) that these errors are amplified by the matrix Z defined by

$$Z(z) = z(I - zB)^{-1}(A - B); \quad z := \lambda h_n.$$

We introduce the *stiff* and *non-stiff amplification matrices* of scheme (5.2), notation $Z_\infty(B)$ and $Z_0(B)$, respectively, by:

$$Z_\infty(B) := \lim_{|z|\to\infty} Z(z) = I - B^{-1}A \quad \text{and} \quad Z_0(B) := \lim_{|z|\to 0} Z(z)/|z| = A - B\,.$$

Choosing $B = A$ would lead to the modified Newton process, for which $Z(z) = 0$ for all z. However, the computation of $Y_n^{(j)}$ now requires the solution of a linear system of dimension sd. For high-dimensional problems this requires a lot of computational effort. Several attempts have been made to reduce these costs by selecting matrices B different from A.

In [CB83], Cooper & Butcher propose the choice $B = P$, where P is a matrix that has a one-point spectrum. By performing a similarity transformation to (5.2) they arrive at the scheme

$$\begin{aligned} PQ &= QL\,, \\ (I - L \otimes h_n J_n)\Delta X_n^{(j+1)} &= -(Q^{-1} \otimes I)R(Y_n^{(j)})\,, \\ Y_n^{(j+1)} &= Y_n^{(j)} + (Q \otimes I)\Delta X_n^{(j+1)}\,, \end{aligned} \tag{5.3}$$

where L and Q are lower triangular and orthogonal matrices, respectively, that define the Schur decomposition of P. Since the diagonal entries of L are equal, implementing (5.3) requires only one LU-decomposition of dimension d.

In [HS91], the authors select $B = D$, where D is a diagonal matrix. Scheme (5.2) is now suitable for implementation on an s processor machine, since the s components of $Y_n^{(j)}$ can be computed independently. The matrix D is constructed such that $\rho(Z_\infty(D)) = 0$, where $\rho(\cdot)$ denotes the spectral radius function. This method was called PDIRK, Parallel Diagonal-implicit Iterated Runge–Kutta.

In Chapter 4, a mixture of the two strategies described above was presented and given the name PTIRK, Parallel Triangularly-implicit Iterated Runge–Kutta. Here, the matrix B was identified with a lower triangular matrix T such that $A = TU$ is the Crout decomposition of A, i.e. U is unit upper triangular. One easily verifies that for this T the stiff amplification matrix $Z_\infty(T)$ is strictly upper triangular. Throughout this chapter, T

will always denote this special lower triangular matrix. This choice of B yields, just like in PDIRK, a stiff amplification matrix that has a zero spectral radius. However, the new strategy leads to an amplification matrix $Z(z)$ that has a much smaller departure from normality than the amplification matrix in PDIRK. Consequently, the amplification after several iterations, i.e. the norm of the powers of $Z(z)$ is now considerably smaller (see Table 4.2). Suppose that all diagonal entries of T are distinct and that the eigenvalue decomposition of T is given by $TQ = QD$, where D is diagonal and Q non-singular. Applying a similarity transformation in an analogous way as in [CB83], we arrive at the scheme

$$\begin{aligned} TQ &= QD, \\ (I - D \otimes h_n J_n)\Delta X_n^{(j+1)} &= -(Q^{-1} \otimes I)R(Y_n^{(j)}), \\ Y_n^{(j+1)} &= Y_n^{(j)} + (Q \otimes I)\Delta X_n^{(j+1)}. \end{aligned} \tag{5.4}$$

It is clear that the s components of $Y_n^{(j)}$ can be computed in parallel. The only additional costs of (5.4) with respect to PDIRK are the appliance of the transformations $(Q \otimes I)$ and $(Q^{-1} \otimes I)$.

In order to ensure the non-singularity of the matrix $(I - D \otimes h_n J_n)$ in (5.4), the positiveness of the diagonal entries of D is required. In Chapter 4 the positiveness of D was proved for $s \leq 5$ and conjectured for $s > 5$. The main scope of this chapter is to prove this conjecture. This will be done in §5.3, using operator theory.

The outline of the rest of the chapter is as follows. §5.2 gives some preliminaries to the conjecture. In §5.4 we prove for $s = 2$, that the choice $B = T$ made in PTIRK is in some sense optimal.

5.2 Preliminaries

The $s \times s$ matrix A belonging to the RK collocation method with abscissa vector c has the form [HW91, p. 82]

$$A = CVRV^{-1},$$

where $C = \text{diag}\{c_1, c_2, \ldots, c_s\}$, $R = \text{diag}\{1, 1/2, \ldots, 1/s\}$ and V is the Vandermonde matrix generated by c, i.e.

$$V = \begin{bmatrix} 1 & c_1 & \ldots & c_1^{s-1} \\ \vdots & \vdots & & \vdots \\ 1 & c_s & \ldots & c_s^{s-1} \end{bmatrix}.$$

Here, the abscissae c_i have to be distinct. In the sequel the abscissae are also supposed to be positive. Without loss of generality, we assume that the RK method is written such that $c_1 < c_2 < \ldots < c_s$. Let $A = TU$ denote the Crout decomposition of A. The diagonal entries t_{kk} of T satisfy the formula (cf. Chapter 4)

$$t_{kk} = \frac{|A_k|}{|A_{k-1}|}, \tag{5.5}$$

where $|A_j|$ denotes the determinant of the j^{th} principal sub-matrix of A and $|A_0| := 1$. From (5.5) we see that the existence of the Crout decomposition immediately follows from the positiveness of t_{kk}.

In Chapter 4 the positiveness of t_{kk}, $k \in \{1, 2, \ldots, s\}$, for $s \leq 5$ was proved in the following way: first it was shown that $|A_1|$ and $|A_s|$ are positive (for general s); then the positiveness of the remaining $|A_2|, \ldots, |A_{s-1}|$ was demonstrated by computing them explicitly; this approach does not lead to a proof for general s.

Another idea is to investigate whether the matrix VRV^{-1} is positive definite. By using the result that every positive definite matrix has an LU-decomposition with positive diagonal entries [GL89, p. 140], the proof of the conjecture would then easily follow, realizing that $T = C\,L$, where L is the lower triangular matrix in the Crout decomposition of VRV^{-1}. However, the following example shows that VRV^{-1} is not always positive definite: If $s = 3$, $c = (1/3, 1/2, 2/3)^{\mathrm{T}}$ and $x = (1, -3, -7)^{\mathrm{T}}$, then $x^{\mathrm{T}}VRV^{-1}x = -11$.

In the following section the proof of the conjecture will be given by considering VRV^{-1} as the matrix of an operator on the space of polynomials of degree less than s with respect to a basis of Lagrange polynomials.

5.3 Proof of the conjecture

THEOREM 5.1 *Let V be the $s \times s$ Vandermonde matrix generated by $c_1, c_2, \ldots, c_s$, where $0 < c_1 < c_2 < \ldots < c_s$, let R be the diagonal matrix* $\operatorname{diag}(1, 1/2, \ldots, 1/s)$. *There exist a lower triangular matrix L, and unit upper triangular matrix U, such that $LU = VRV^{-1}$. The diagonal entries of L are positive.*

Notice that from this theorem it immediately follows that for any $s \times s$ RK collocation matrix A with positive distinct abscissae, there exists a lower triangular matrix T with positive diagonal entries such that $Z_\infty(T)$ is strictly upper triangular, by setting $T = C\,L$.

PROOF Let $\mathbb{P}_s$ be the s-dimensional linear space of polynomials of degree less than s with real coefficients, and $\mathcal{C}$ the canonical basis for $\mathbb{P}_s$, i.e.

$$\mathcal{C} = \{1, x, \ldots, x^{s-1}\}.$$

Define the operator $H : \mathbb{P}_s \to \mathbb{P}_s$ by $H(p) = q$ where q is defined by

$$q(x) = \frac{1}{x} \int_0^x p(t)\,\mathrm{d}t\,.$$

We use the notation $\operatorname{mat}(H)_{\mathcal{C}}$ for the matrix of the operator H with respect to the basis $\mathcal{C}$. It can be easily verified that

$$\operatorname{mat}(H)_{\mathcal{C}} = R\,.$$

We denote the k^{th} Lagrange polynomial with respect to $c_1, c_2, \ldots, c_s$ by l_k:

$$l_k(x) = \prod_{i \neq k} \frac{x - c_i}{c_k - c_i} \quad ; \quad k \in \{1, 2, \ldots, s\}.$$

Notice that l_k is of degree $s-1$ and thus element of $\mathbb{P}_s$. The Lagrange polynomials define also a basis for $\mathbb{P}_s$, which will be denoted by $\mathcal{L}$:

$$\mathcal{L} = \{l_1, l_2, \ldots, l_s\}.$$

We write $\mathcal{C}_\mathcal{L}$ for the matrix that expresses the canonical basis in the Lagrange basis. Since for every $m \in \{0, 1, \ldots, s-1\}$ the equality

$$x^m = c_1^m l_1 + c_2^m l_2 + \cdots + c_s^m l_s$$

should hold, it can be seen that $\mathcal{C}_\mathcal{L} = V$. Consequently, the matrix of the operator H with respect to the basis $\mathcal{L}$ is given by

$$\mathrm{mat}(H)_\mathcal{L} = \mathcal{C}_\mathcal{L} \cdot \mathrm{mat}(H)_\mathcal{C} \cdot \mathcal{C}_\mathcal{L}^{-1} = VRV^{-1} =: B\,.$$

If $\big(H(l_k)\big)_\mathcal{L}$ denotes the image under H of l_k with respect to the basis $\mathcal{L}$, then

$$\big(H(l_k)\big)_\mathcal{L} = Be_k = \begin{pmatrix} \beta_{1k} \\ \vdots \\ \beta_{nk} \end{pmatrix},$$

where e_k is the k^{th} canonical basis vector of $\mathbb{R}^s$ and $(\beta_{ij}) = B$.
We claim that $\beta_{11} > 0$. To see this, notice that $H(l_1)$ is a polynomial with coefficient β_{11} in the direction of l_1. Since $l_k(c_1) = 0$ for $k > 1$, it is clear that

$$\big(H(l_1)\big)(c_1) = \beta_{11}.$$

With respect to the value of l_1 in zero, we observe that $l_1(c_1) = 1$, and that all its roots are to the right of c_1; therefore l_1 is positive on $[0, c_1]$, which implies

$$\big(H(l_1)\big)(c_1) = \frac{1}{c_1}\int_0^{c_1} l_1(t)\,\mathrm{d}t > 0.$$

Consequently, $\beta_{11} > 0$.

It is now possible to define

$$v_{1k} := -\frac{\beta_{1k}}{\beta_{11}} \quad ; \quad k \in \{2, \ldots, s\}.$$

From this definition it follows that, for $k > 1$,

$$\big(H(l_k + v_{1k}l_1)\big)_\mathcal{L} = B(e_k + v_{1k}e_1) = B\begin{pmatrix} v_{1k} \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \beta_{2k}^{(1)} \\ \vdots \\ \beta_{nk}^{(1)} \end{pmatrix}.$$

Assuming $\beta_{22}^{(1)} \neq 0$, we are able to define

$$v_{2k} := -\frac{\beta_{2k}^{(1)}}{\beta_{22}^{(1)}} \quad ; \quad k \in \{3, \ldots, s\},$$

such that

$$\left(H(l_k + v_{2k} l_2 + v_{1k} l_1)\right)_{\mathcal{L}} = B \begin{pmatrix} v_{1k} \\ v_{2k} \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \beta_{3k}^{(2)} \\ \vdots \\ \beta_{nk}^{(2)} \end{pmatrix}.$$

Continuing this procedure, we finally arrive at

$$\left(H(\sum_{i=1}^{k} v_{ik} l_i)\right)_{\mathcal{L}} = B u_k = r_k \,,$$

where

$$v_{ik} = \begin{cases} -\frac{\beta_{ik}^{(i-1)}}{\beta_{ii}^{(i-1)}} & \text{for} \quad i < k \,, \\ 1 & \text{for} \quad i = k \,, \end{cases}$$

(defining $\beta_{ij}^{(0)} = \beta_{ij}$) and u_k and r_k are vectors defined by

$$u_k = \begin{pmatrix} v_{1k} \\ \vdots \\ v_{k-1,k} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{and} \quad r_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \beta_{kk}^{(k-1)} \\ \vdots \\ \beta_{nk}^{(k-1)} \end{pmatrix}.$$

If we can show that $\beta_{kk}^{(k-1)} > 0$ for $k \in \{2, 3, \ldots, s\}$, we have demonstrated that the procedure outlined above can be carried out. By observing that u_k and r_k are columns of matrices $\widetilde{U}$ and L, respectively, for which the relation $B\widetilde{U} = L$ holds, we then have proved Theorem 5.1 using U for $\widetilde{U}^{-1}$.

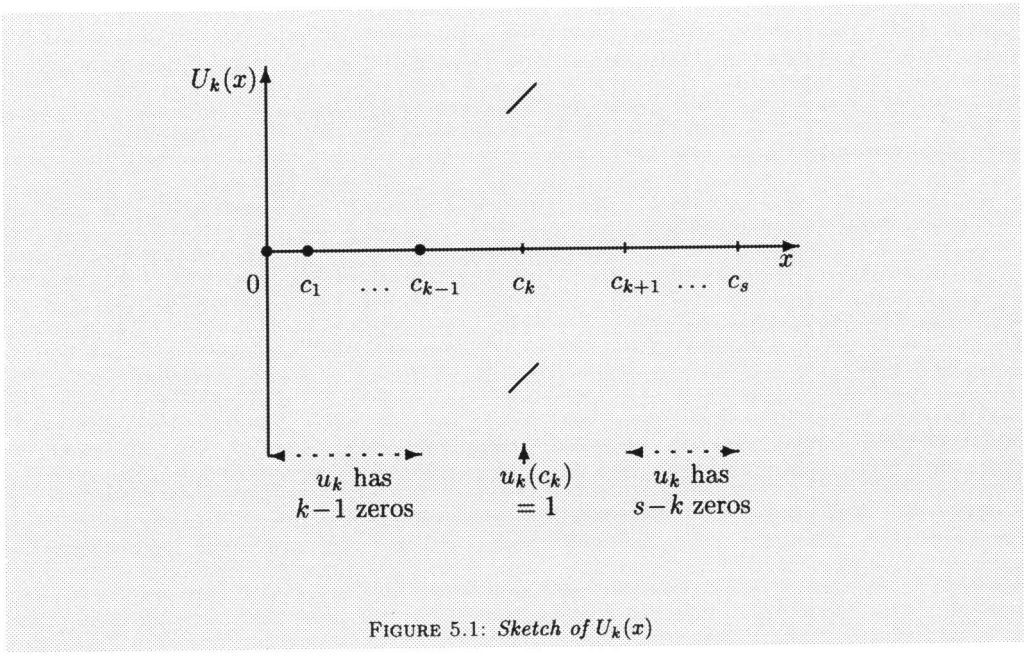

FIGURE 5.1: *Sketch of* $U_k(x)$

The vectors u_k and r_k can be considered as polynomials in $\mathbb{P}_s$ with respect to the basis $\mathcal{L}$. Moreover, r_k is the image of u_k under the operator H:

$$H(u_k) = r_k\,.$$

Since $r_k(c_k) = \beta_{kk}^{(k-1)}$, we have to prove that $r_k(c_k) > 0$. We define the polynomial U_k of degree $s+1$ by

$$U_k(x) = \int_0^x u_k(t)\,\mathrm{d}t\,.$$

Notice that $U_k(0) = 0$ and, for $x > 0$, the sign of r_k equals the sign of U_k (the latter holds since $U_k = xr_k$). Since $l_k(c_i) = 0$ for $i < k$ and r_k has only components in the direction of l_j with $j \geq k$, we see that $r_k(c_i) = 0$ for $i < k$ and consequently

$$U_k(c_i) = 0 \quad \text{for} \quad i < k.$$

This means that u_k (being the derivative of U_k) has $k-1$ zeros in the interval $(0, c_{k-1})$. All components of u_k in the direction of the last $s-k$ Lagrange polynomials are zero. Consequently, $u_k(c_i) = 0$ for $i > k$, so that u_k has $s-k$ zeros in the interval $[c_{k+1}, c_k]$.

We now consider 2 cases (see also Figure 5.1):

$$u_k(c_{k-1}) \;>\; 0\,, \tag{5.6}$$

$$u_k(c_{k-1}) \;<\; 0\,. \tag{5.7}$$

Remark that, since all c_i are distinct, U_k has a single zero in c_{k-1}, so that the situation $u_k(c_{k-1}) = 0$ does not arise. Suppose that (5.7) holds. Since $u_k(c_k) = 1$, the polynomial u_k should have a zero in the interval (c_{k-1}, c_k). In that case, u_k has $(k-1)+(s-k)+1 = s$ zeros. However, the degree of u_k is only $s-1$, proving that only situation (5.6) can occur, and $u_k > 0$ on (c_{k-1}, c_k).

From $U_k(c_{k-1}) = 0$, it now follows that $U_k(c_k) > 0$. Since r_k has the same sign as U_k, we have proved the theorem. □

5.4 Is PTIRK optimal?

In this section we investigate the optimality of the matrix T in PTIRK. Since the number of parameters becomes too large to handle conveniently for $s > 2$, we restrict ourselves here to methods with 2 implicit stages, i.e. $s = 2$.

In the class of lower triangular matrices, T is optimal in the sense that it leads to the smallest stiff amplification matrix measured in the infinity norm:

THEOREM 5.2 *If L is a 2×2 lower triangular matrix, then*

$$\|Z_\infty(L)\|_\infty \geq \|Z_\infty(T)\|_\infty .$$

PROOF Write $L^{-1} = (l_{ij})$ with $l_{12} = 0$. Then

$$Z_\infty(L) = \begin{pmatrix} 1 + \frac{l_{1,1} c_1 (-2\,c_2 + c_1)}{2\,(c_2 - c_1)} & \frac{l_{1,1} {c_1}^2}{2\,(c_2 - c_1)} \\ * & * \end{pmatrix}.$$

Define for $x > 0$:

$$g(x) = \left| 1 + \frac{c_1\,(-2\,c_2 + c_1)}{2\,(c_2 - c_1)}\, x \right| + \frac{c_1^2}{2\,(c_2 - c_1)}\, x .$$

Then $g(x) \geq g(x_{\min}) = c_1/(2\,c_2 - c_1)$, where $x_{\min} = 2\,(c_2 - c_1)/(c_1\,(2\,c_2 - c_1))$. Since $\|Z_\infty(T)\|_\infty = g(x_{\min})$, it follows that $\|Z_\infty(L)\|_\infty \geq \|Z_\infty(T)\|_\infty$. □

For two well-known stiffly accurate RK methods with 2 implicit stages, it is possible to show that in the class of lower triangular matrices that lead to a 'small' stiff amplification matrix, T is optimal in the sense that it has the smallest non-stiff amplification matrix, again measured in the infinity norm:

THEOREM 5.3 *If L is a 2×2 lower triangular matrix with the property that $\rho(Z_\infty(L)) = 0$, then, for the 2-stage RadauIIA, and the 3-stage Lobatto IIIA method,*

$$\|Z_0(L)\|_\infty \geq \|Z_0(T)\|_\infty .$$

PROOF Write $A = (a_{ij})$ and $L = (l_{ij})$ with $l_{12} = 0$. Then $\|Z_0(L)\|_\infty = \max\{m_1, m_2\}$, where m_1 and m_2 are given by

$$m_1 = |a_{11} - l_{11}| + |a_{12}| \quad \text{and} \quad m_2 = |a_{21} - l_{21}| + |a_{12} - l_{22}|\,.$$

Let J be the interval such that if $l_{11} \notin J$, then $m_1 > \|Z_0(T)\|_\infty$. Notice that J only depends on c. From $\sigma(Z_\infty(L)) = 0$ it follows that $\operatorname{trace}(Z_\infty(L)) = \det(Z_\infty(L)) = 0$. Using these two equations, it is possible to express l_{21} and l_{22}, and thus m_2, in l_{11}. We have to proof that for $l_{11} \in J$, $m_2 \geq \|Z_0(T)\|_\infty$. We treat the two methods separately.

RadauIIA: $c = (1/3, 1)^{\mathrm{T}}$, $\|Z_0(T)\|_\infty = 3/20$, $J = [7/20, 29/60]$, and

$$m_2(l_{11}) = \left|\frac{3}{4} + \frac{-24\,l_{1,1} + 5 + 18\,l_{1,1}^2}{6\,l_{1,1}}\right| + \left|\frac{1}{4} - \frac{1}{6\,l_{1,1}}\right|.$$

It can be verified that $\min\limits_{l_{11} \in J} \{m_2(l_{11})\}) = m_2(t_{11}) = 3/20$.

LobattoIIIA: $c = (0, 1/2, 1)^{\mathrm{T}}$, $\|Z_0(T)\|_\infty = 1/12$, $J = [7/24, 3/8]$, and

$$m_2(l_{11}) = \left|\frac{2}{3} + \frac{-12\,l_{1,1} + 2 + 12\,l_{1,1}^2}{3\,l_{1,1}}\right| + \left|\frac{1}{6} - \frac{1}{12\,l_{1,1}}\right|.$$

The reader is invited to check that $\min\limits_{l_{11} \in J} \{m_2(l_{11})\} = m_2(t_{11}) = 1/12$.

□

Chapter 6

Sparse matrix solvers

Abstract The use of implicit methods for numerically solving stiff systems of differential equations requires the solution of systems of non-linear equations. Normally these are solved by a Newton-type process, in which we have to solve systems of linear equations. The Jacobian of the derivative function determines the structure of the matrices of these linear systems. Since it often occurs that the components of the derivative function only depend on a small number of variables, the system can be considerably sparse. Hence, it can be worth the effort to use a sparse matrix solver instead of a dense LU-decomposition. This chapter reports on experiences with the direct sparse matrix solvers MA28 by Duff [Duf77], Y12M by Zlatev et al. [ZWS81] and one special-purpose matrix solver, all embedded in the parallel ODE solver PSODE by Sommeijer [Som93].

6.1 Introduction

For solving the stiff initial value problem (IVP)

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \leq t \leq t_{\text{end}}, \tag{6.1}$$

one of the most powerful methods is an implicit Runge–Kutta (RK) scheme. However, in such a method we have to solve a system of non-linear equations of dimension sd in every time step. Here, s is the number of stages. This may require a lot of computational effort and for this reason implicit RKs have not been very popular on sequential computers. On parallel computer architectures the costs can be reduced significantly. Several ways of doing this are described in [Bel87, BVZ90, Bur93a, Cha93, Ore93]. In this chapter we will focus on PSODE (Parallel Software for ODEs), described in [Som93]. PSODE is an implementation of a Radau IIA method with a Newton-type iteration process, which reduces the non-linear systems to sequences of linear systems of dimension d. Most time in PSODE is spent on the solution of these linear systems, using a dense LU-decomposition, followed by forward-back substitutions. For problems arising in practice it often occurs that the components of the derivative function only depend on a small number of variables. Such problems lead to linear systems of which the matrix is sparse. The goal of this chapter is to reduce the linear algebra costs for such problems by using a sparse matrix solver.

We remark that the underlying method of PSODE is in terms of linear systems so similar to the methods derived in Chapter 4 and 7, that the results on sparse matrix solvers presented here can be directly applied to codes based on these methods. This holds in particular for the code PSIDE, which is based on the method in Chapter 7 and is the subject of Chapters 11–12.

The outline of this chapter is as follows. §6.2 briefly describes PSODE. §6.3 presents the off-the-shelf sparse matrix solvers MA28 [Duf77] and Y12M [ZWS81] and a sparse matrix solver that is designed especially for ODE solvers. An analysis of the influence of the errors made in the numerical solution of the linear systems is given in §6.4. Finally, in §6.5, numerical experiments give insight in the behavior of these matrix solvers.

6.2 The parallel ODE solver PSODE

In the following the m^{th} canonical basis vector of $\mathbb{R}^s$ will be denoted by e_m, and the $m \times m$ identity matrix by I_m.

PSODE is a code based on a parallel method for numerically solving problems of type (6.1). It is based on the implicit RK method Radau IIA, for which the number of stages s equals 4. Denoting the Radau IIA matrix by A, we have to solve in every time step the sd-dimensional system of non-linear equations

$$Y_n = (\mathbb{1} \otimes I_d)y_{n-1} + h_n(A \otimes I_d)F(Y_n). \tag{6.2}$$

Here, Y_n is the sd-dimensional stage vector $(Y_{n,i})$ containing approximations to $y(t_{n-1} + c_i h_n)$, $i = 1, \ldots, s$, in which $c = (c_1, \ldots, c_s)^{\mathrm{T}}$ is the abscissa vector, $\mathbb{1}$ is the s-dimensional unit vector $(1, \ldots, 1)^{\mathrm{T}}$, y_{n-1} is the approximation to the step point value $y(t_{n-1})$, h_n is the stepsize $t_n - t_{n-1}$ and $F(Y_n)$ contains the derivative values $f(Y_{n,i})$. Since $c_s = 1$ for Radau IIA methods, Y_n contains y_n:

$$y_n = (e_s^{\mathrm{T}} \otimes I_d)Y_n.$$

Solving (6.2) by a modified Newton process would yield a sequence of iterates $Y_n^{(0)}$, $Y_n^{(1)}$, $Y_n^{(2)}, \ldots$ defined by

$$Y_n^{(0)} = \text{given by some predictor formula,} \tag{6.3}$$

$$(I_{sd} - h_n(A \otimes J_n))\Delta Y_n^{(j+1)} = -R(Y_n^{(j)}), \qquad j = 0, 1, 2, \ldots, \tag{6.4}$$

where J_n is an approximation to the Jacobian of f in (t_{n-1}, y_{n-1}), $\Delta Y_n^{(j+1)} = Y_n^{(j+1)} - Y_n^{(j)}$ and $R(Y_n^{(j)})$ denotes the residual of $Y_n^{(j)}$ with respect to (6.2), i.e.

$$R(Y_n^{(j)}) = Y_n^{(j)} - (\mathbb{1} \otimes I_d)y_{n-1} - h_n(A \otimes I_d)F(Y_n^{(j)}).$$

The process (6.3), (6.4) requires the solution of linear systems of dimension sd. Since the approximation J_n does not vary during a time step, we are dealing in every time step with a sequence of linear systems with the same matrix. In practice we often keep the approximation of the Jacobian and the stepsize h_n constant over a number of time steps. Hence, the number of linear systems with the same matrix is frequently a multiple of the number of Newton iterations per time step. This explains our bias to use a direct matrix solver instead of e.g. an iterative Krylov subspace method; in the latter case we would have to rebuild the Krylov subspace for every new right-hand side, whereas with an LU-decomposition, for every new right-hand side, only the forward-back substitutions

have to be performed. However, in an iterative approach, it may be possible to exploit the fact that the linear systems are related; this will be subject of future research.

The costs of an LU-decomposition of the matrix in (6.4) are $\mathcal{O}(s^3d^3)$. These costs can be reduced by replacing A by a matrix $D = \text{diag}\{d_1, \ldots, d_s\}$. The linear system of dimension sd is now decoupled into s systems of dimension d:

$$(I_d - h_n d_i J_n))\Delta Y_{n,i}^{(j+1)} = -(e_i^{\mathrm{T}} \otimes I_d)R(Y_n^{(j)}); \quad i = 1, 2, \ldots, s; \quad j = 0, 1, 2, \ldots \tag{6.5}$$

These systems can be solved in parallel if s processors are available: Every processor makes an LU-decomposition of $I_d - h_n d_i J_n$ and performs the forward-back substitutions on the right-hand side $-(e_i^{\mathrm{T}} \otimes I_d)R(Y_n^{(j)})$. The sequential costs on s processors are thus $\mathcal{O}(d^3)$. Notice that it is possible to compute the components of $R(Y_n^{(j)})$ in parallel too: Every processor computes $f(Y_{n,i}^{(j)})$, broadcasts the result to the other $s-1$ processors and receives the remaining part of $F(Y_n^{(j)})$ from the other processors.

In PSODE the matrix D is chosen such that the stiff components of the iteration errors are strongly damped (see [HS91, HS93]). We shall refer to (6.3), (6.5) as a *simplified Newton process*. PSODE uses this simplified Newton process to solve (6.2) together with strategies for determining when the Jacobian should be re-evaluated, when a new LU-decomposition should be made and how many iterations should be performed in (6.5). A stepsize control is also included. For details on implementation, we refer the reader to [Som93].

6.3 Sparse matrix solvers

A general direct solver for non-symmetric, sparse linear systems is in most cases based on the Gaussian Elimination process (GE). The main feature that characterizes a direct sparse matrix solver is the pivoting strategy. A balance has to be found between *stability pivoting* and *sparsity pivoting*. Stability pivoting means reordering the matrix to obtain pivots that are relatively large, so that GE becomes numerically stable. The aim of sparsity pivoting is to find a reordering of the matrix such that the number of operations and the fill-in of the reordered matrix are kept small. Here, the fill-in of a matrix M after reordering is defined as follows. Suppose P_1 and P_2 are permutation matrices such that $P_1 M P_2$ denotes the reordered matrix. If

$$P_1 M P_2 = LU,$$

where L is lower triangular with $\text{diag}\{L\} = (1, 1, \ldots, 1)$ and U is upper triangular, then the fill-in of $P_1 M P_2$ is the number of nonzeros in the strictly lower triangular part of L + the number of nonzeros in U − the number of nonzeros in M. A well-known strategy to keep the fill-in small is to use the *Markowitz criterion* [Mar57]. Suppose that the first k steps of GE have been performed. Let $r_i^{(k)}$ and $c_j^{(k)}$ be the number of nonzero entries

in the i^{th} row and j^{th} column of the remaining $(n-k) \times (n-k)$ sub-matrix, respectively. Then the Markowitz criterion selects as pivot the nonzero element

$$m_{ij}^{(k)} \text{ with } i,j \text{ such that } (r_i^{(k)} - 1)(c_j^{(k)} - 1) \text{ is minimal.}$$

Notice that by this definition a row or column with only one nonzero entry automatically delivers this entry as pivot. Unfortunately, the Markowitz ordering does not always produce the best ordering. On the other hand, the problem of finding the reordering that really minimizes the fill-in is NP-complete [DER86]. In the sequel we will refer to $(r_i^{(k)} - 1)(c_j^{(k)} - 1)$ as the *Markowitz number of* $m_{ij}^{(k)}$, where $M^{(k)} = (m_{ij}^{(k)})$ denotes the matrix M after $k-1$ steps of GE.

Another categorization of pivoting strategies distinguishes between regions of the matrix in which the pivot is to be found. The case where the pivot of the k^{th} step in GE is determined in the last $n-k+1$ entries of the k^{th} column is referred to as *partial* pivoting. If the pivot is chosen on the main diagonal, we speak of *diagonal* pivoting. *Complete* pivoting means scanning the whole sub-matrix $M(k:n, k:n)$ to select the pivot. In a straightforward way one derives combinations of several pivoting strategies. For example, diagonal Markowitz pivoting means selecting in the k^{th} step of GE the nonzero element $m_{ii}^{(k)}$, of which the Markowitz number is minimal.

6.3.1 MA28

MA28 is a set of Fortran subroutines for sparse unsymmetric linear equations. This code by Duff [Duf77] is part of the Harwell Subroutine Library, which is licensed. However, one may use this code for research purposes. The user can set a parameter u to control bias towards stability pivoting or sparsity pivoting: $u = 1.0$ gives partial stability pivoting, while $u = 0.0$ minimizes the fill-in without checking the magnitude of the pivots at all. The sparsity pivoting is performed by means of the Markowitz criterion. For values of $u \in (0,1)$ a stability control is added. The user supplies the sparse matrix in a 1-dimensional array containing the nonzeros. The row and column indices in the sparse matrix are stored in two 1-dimensional integer arrays. Since MA28 performs the LU-decomposition and the forward-back substitutions in separate subroutines, it is easy to solve sequences of linear systems with the same matrix by performing only 1 decomposition.

In the version of MA28 dated 1 January 1984, which we used, common blocks that communicate data between the 17 internal subroutines of the package, complicate parallel implementation of the code. One may get around this problem by using more up-to-date versions.

6.3.2 Y12M

Y12M is a package of Fortran subroutines for the same purpose as MA28. It was developed at the Regional Computing Centre at the University of Copenhagen (RECKU) by Zlatev et al. One can obtain the code from Netlib [DG87]. The complete documentation is in [ZWS81]. Although Y12M is similar to MA28, the influence of the user on

the choice of the pivoting strategy is different. Although the code selects by itself the mixture between sparsity and stability pivoting, the user can decide where the pivot is to be selected. He can restrict the pivots to the diagonal and it is also possible to choose in how many rows that have least number of non-zero elements the pivotal search is carried out. Again the underlying sparsity pivoting strategy is based on the Markowitz criterion.

6.3.3 Special purpose solver

Our first experiments with Y12M and MA28 suggested that for our test problems the use of stability pivoting had hardly any influence. The experience that stability pivoting is seldom required for solving stiff ODEs was also reported by others; see [JT94, Hin83, VBLS95, DER86]. In the next section we give a heuristic explanation of this phenomenon. Therefore, we also implemented a special purpose matrix solver without stability pivoting. In this solver, we use the following strategy:

1. Use only diagonal sparsity pivoting in order to reduce the fill-in of the matrix $I_d - h_n d_i J_n$.
2. Compute the fill-in of the reordered matrix and add the elements that will be made nonzero to the sparse data structure.
3. Perform an Incomplete LU-decomposition (ILU) on the augmented reordered matrix of step 2.
4. Perform the forward-back substitutions with the reordered right-hand sides.

We reorder the matrix with the diagonal Markowitz strategy. Remember that the sparsity structure of $I_d - h_n d_i J_n$ remains constant over the integration interval, so that step 1 and 2 have to be done only once. Since the magnitude of the entries is not involved, this is a symbolic operation. We programmed step 1 and 2 in Maple [CGG+91].

Step 3 and 4 are performed by modified versions of subroutines of the Sparse Linear Algebra Package (SLAP) that perform ILU as preconditioner. SLAP is written by Greenbaum and Seager (with contributions of several other authors) and is available from Netlib [DG87]. It uses the compressed row/column format. Notice that the input for these subroutines does not only contain the nonzero elements of the matrix $I_d - h_n d_i J_n$, but also the zero elements that will be made nonzero by the GE process. Consequently, the ILU performed in step 3 is algebraically equivalent with a complete LU-decomposition. In the sequel, we will refer to this sparse matrix solver as Special Purpose Solver (SPS). Remark that both MA28 and Y12M do not allow the pivoting strategy followed in SPS.

6.4 Error analysis

In this section we investigate how the omission of stability pivoting in solving linear systems arising from ODEs, that are solved numerically with the use of a Newton-type process, influences the numerical solution to the ODE.

In the sequel we use the short hand notations M for $I_d - h_n d_i J_n$, the short form x_j for $Y_{n,i}^{(j)}$ and $r(x_j)$ for $-(e_i^{\mathrm{T}} \otimes I_d)R(Y_n^{(j)})$. Denoting the update $x_{j+1} - x_j$ by u_{j+1}, the linear system (6.5) takes the form

$$Mu_{j+1} = r(x_j). \tag{6.6}$$

Other implicit ODE solvers, like e.g. codes based on Backward Differentiation Formulas, lead to linear systems for which the remainder of this section holds as well.

First note that neither the matrix M (if non-singular) nor its decomposition have influence on the *solution* of the Newton process; they can only affect the rate of convergence to this solution.

Assume that (6.6) can be solved using GE with some pivoting strategy. Hence, the inverse matrix M^{-1} exists. The omission of stability pivoting may lead to an accidental breakdown, if a diagonal element of M becomes zero. However, a change of stepsize will cure this breakdown. If the convergence of the Newton process stagnates because of too small pivots, then the integrator would detect this and restrict the stepsize. The matrix M becomes more diagonal dominant and the pivots using no pivoting or diagonal pivoting would now be relatively larger. Hence, GE becomes more stable. On the other hand, for efficiency reasons, we want the stiff solver to use large steps. Experiments suggest that the increase of step rejections due to the omission of stability pivoting is very modest.

Let us now look in more detail to the error propagation in the Newton process. Assuming that x is such that $r(x) = 0$, that no error is made when solving the linear systems and defining the Newton error by $\delta_j := x_j - x$, we arrive at

$$\begin{aligned} \delta_{j+1} &= \delta_j + M^{-1}(r(x_j) - r(x)) \\ &= (I + M^{-1}Q(x))\delta_j + \text{ higher order terms} \\ &= P^{j+1}\delta_0 + \text{ higher order terms.} \end{aligned} \tag{6.7}$$

Here, $Q(x)$ is the Jacobian of $r(x)$ and $P := I + M^{-1}Q(x)$. However, due to the omission of stability pivoting, we compute instead of $x_1, x_2, \ldots$ the sequence $\widetilde{x}_1, \widetilde{x}_2, \ldots$, that satisfies

$$(M + E)\widetilde{u}_{j+1} = r(\widetilde{x}_j), \tag{6.8}$$

where $\widetilde{u}_{j+1} := \widetilde{x}_{j+1} - \widetilde{x}_j$. Notice that we neglected here other rounding errors than those arising in the LU-factorization of M. Defining the linear system error in the j^{th} Newton update by $\epsilon_j := \widetilde{u}_j - \widehat{u}_j$, where $\widehat{u}_j$ is the solution of $M\widehat{u}_j = r(\widetilde{x}_{j-1})$, we arrive at

$$\epsilon_j = -M^{-1}E\,\widetilde{u}_j. \tag{6.9}$$

Combining the formulas (6.8), (6.9), (6.7) and $\widetilde{\delta}_0 = \delta_0$, yields a formula for the Newton errors in which the linear system errors are taken into account too, described by $\widetilde{\delta}_j := \widetilde{x}_j - x$:

$$\widetilde{\delta}_j = \delta_j + \sum_{k=1}^{j} P^{j-k}\epsilon_k + \text{ higher order terms.} \tag{6.10}$$

In the sequel, we denote the spectrum and spectral radius of any matrix X by $\sigma(X)$ and $\rho(X)$.

The formulas above lead us to several indications why omitting the stability pivoting works well. First of all, for dissipative systems, it holds that

$$\sigma(J_n) \in \{z \in \mathbb{C} \mid \mathrm{Re}(z) \leq 0\},$$

and consequently, since the diagonal entries $d_i > 0$,

$$\sigma(M) \in \{z \in \mathbb{C} \mid \mathrm{Re}(z) \geq 1\} \quad \text{and} \quad \rho(M^{-1}) \leq 1.$$

This is a necessary (although not sufficient) condition for M to damp the error matrix E in (6.9).

Secondly, for $h \to 0$, the matrix M becomes increasingly diagonal dominant. This means $\|E\| \to 0$. On the other hand, the situation $h \to \infty$ is usually initiated by an ODE-solution that tends to a steady state. Here we expect that $\forall j$, $\widetilde{u}_j \to 0$. In both situations, $\forall k$, $\epsilon_k \to 0$, so $\widetilde{\delta}_j \to \delta_j$.

Our last argument is based on formulas (6.9) and (6.10). Normally, Newton iterates are monotonically decreasing:

$$\|\widetilde{u}_k\| < \|\widetilde{u}_j\| \quad \text{for} \quad k > j.$$

Together with formula (6.9) this tells us that it is likely that the error matrix E has less influence on ϵ_k as k increases. However, the contribution of ϵ_k in $\widetilde{\delta}_j$ is by means of (6.10) multiplicated by P^{j-k} and thus more damped for small k, since P is a contracting operator if the Newton process converges.

Although we did not give a rigorous proof that the omission of stability pivoting in the solution process of the non-linear systems arising from ODEs is harmless, we showed in the above that at least a number of necessary conditions is fulfilled.

6.5 Numerical experiments

6.5.1 Test problems

To test how the sparse matrix solvers perform in PSODE we consider 3 stiff test problems. The first one comes from circuit analysis and describes a ring modulator. It is of dimension 15. Our second test problem has dimension 20 and is the chemical part of an air pollution model. The last problem is the EMEP MSC-W ozone chemistry model of dimension 66. For a more detailed description of these problems we refer the reader to [LSV96].

In order to see the effect of an increasing problem size on the performance of the sparse matrix solvers we 'cascade' the problems m times as follows. If the original test problems are of the form

$$\widehat{y}'(t) = \widehat{f}(\widehat{y}), \qquad \widehat{y}(0) = \widehat{y}_0, \qquad \widehat{y} \in \mathbb{R}^{\hat{d}}, \qquad t_0 \leq t \leq t_{\mathrm{end}},$$

TABLE 6.1: *Sparsity characteristics for the three test problems.*

problem ('m times cascaded')	ring modulator	pollution problem	EMEP problem
nonzero ratio	$55/(225m)$	$86/(400m)$	$496/(66^2m)$
fill-in before reordering	$58m$	$176m$	$2166m$
fill-in after diagonal Markowitz	12, 24, 36, 120	9, 17, 24, 82	87, 171, 260, 861
Sparsity structure symmetric?	yes	no	no

then the resulting 'cascaded' problems are of form (6.1), where f is defined by

$$f(y) = \begin{pmatrix} \widehat{f}(\widehat{y}) \\ \vdots \\ \widehat{f}(\widehat{y}) \end{pmatrix}, \quad y = \begin{pmatrix} \widehat{y} \\ \vdots \\ \widehat{y} \end{pmatrix}, \quad y_0 = \begin{pmatrix} \widehat{y}_0 \\ \vdots \\ \widehat{y}_0 \end{pmatrix},$$

and $d = m\widehat{d}$.

Information on the sparsity of the matrix $I_d - h_n d_i J_n$ is listed in Table 6.1. As usual, we define the *nonzero ratio* of a matrix to be the number of nonzero entries divided by the total number of entries. In the table, the fill-in before and after reordering with the diagonal Markowitz strategy is specified for $m \in \{1, 2, 3, 10\}$. For the pollution and EMEP problem we see that the fill-in after reordering does not depend linearly on m. The reason for this is that from the elements with the same Markowitz number the last one is chosen as pivot. Consequently, the diagonal blocks of the Jacobian of the 'm times cascaded' problem are not necessarily treated identically by the reordering algorithm.

6.5.2 Numerical results

Firstly, we experimented with PSODE using Y12M, MA28 and SPS on a one-processor machine in order to compare the matrix solvers mutually and to see the influence of an increasing nonzero ratio of the Jacobian on the performance. We also listed the results of the dense matrix solver of LINPACK (the routines `dgefa` and `dgesl`, for computing the LU-factors and forward-back substitutions, respectively). The Tables 6.2, 6.3 and 6.4 show the results of solving the three test problems. A few remarks with respect to these tables should be made.

- The numerical experiments were done on a Silicon Graphics *Indy* workstation (100 MHz R4000SC), using 64-bit arithmetic. 'CPU' refers to the CPU time in seconds to solve a test problem on this machine.
- m denotes the number of times that the problem is cascaded.
- For MA28, the pivoting parameter u was set equal to 0.5. Other settings of u did not yield significantly better results.

TABLE 6.2: *Results on ring modulator.*

	m	CPU	scd	steps	J-eval	LU-dec	f-eval
LINPACK	1	24.40	4.20	3129	537	2393	19247
	2	67.13					
	3	130.52					
	10	1145.44					
MA28	1	30.57	4.46	3188	568	2462	19784
	2	59.41	4.06	3155	554	2434	19488
	3	96.21					
	10	306.12					
Y12M	1	24.82	4.78	3173	552	2437	19540
	2	50.13	4.04	3165	552	2452	19486
	3	75.86	4.64	3146	548	2443	19408
	10	258.69	4.31	3159	557	2458	19446
SPS	1	14.35	4.49	3159	550	2428	19390
	2	28.16	4.73	3160	554	2424	19481
	3	42.81	4.62	3131	541	2434	19337
	10	153.38	4.53	3150	551	2418	19422

- For Y12M most parameters were set to their default values. Only the drop tolerance was valued 10^{-14} (using the default value 10^{-12}, a breakdown occurred in the pollution problem). Other choices for the parameters that determine the pivoting strategy, did not improve the results considerably.

- The integration statistics of PSODE are given by

  scd denoting the minimum number of significant correct digits in the numerical solution in the end point, that is:

  $$\text{scd} := \min_{i\in\{1,\ldots,d\}} \left\{ -log_{10} \left| \frac{\widetilde{y}_i(t_{\text{end}}) - y_i(t_{\text{end}})}{y_i(t_{\text{end}})} \right| \right\},$$

  where $\widetilde{y}_i(t_{\text{end}})$ and $y_i(t_{\text{end}})$ denote the i^{th} component of the numerical and true solution in the end point, respectively, and d is the dimension of the problem. Since the exact solution to the problems is not known, the numerical solution was compared with the output of a very accurate run.

  steps refers to the number of time steps (including rejected steps).

  J-eval is the number of evaluations of the analytical Jacobian of the function f.

  LU-dec denotes the number of 'sequential' LU-decompositions of matrices of the form $I_d - h_n d_i J_n$.

  f-eval is the number of 'sequential' evaluations of the function f.

  Here, 'sequential' means that operations on the four stages at the same time step, which can be done in parallel, are counted as one.

TABLE 6.3: *Results on pollution problem.*

	m	CPU	scd	steps	J-eval	LU-dec	f-eval
LINPACK	1	0.59	6.11	46	10	44	299
	2	1.80					
	3	3.68					
	10	37.80					
MA28	1	0.67	6.11	46	10	44	299
	2	1.32					
	3	1.92					
	10	6.42					
Y12M	1	0.54	6.11	46	10	44	299
	2	1.14					
	3	1.62					
	10	5.53					
SPS	1	0.29	6.11	46	10	44	299
	2	0.55					
	3	0.85					
	10	2.98					

- The user of PSODE has to supply the error tolerance, a safe upperbound for $\|y(t)\|_\infty$ on the whole integration interval, and the initial stepsize. For the ring modulator and the pollution problem we set them equal to 10^{-4}, 1 and 10^{-7}, respectively, for the EMEP problem to 10^{-2}, 10^{16} and 1.

- Blank entries in the tables are identical to corresponding entries in the adjacent upper row.

We nicely see that the CPU-timings for the sparse matrix solvers are $\mathcal{O}(m)$, whereas for the dense solver of LINPACK they are superlinear. For these 3 test problems it begins to pay off to use MA28 or Y12M instead of LINPACK for nonzero ratios of less than about 20–25%. For SPS this maximum nonzero ratio is even somewhat larger. SPS performs about 1.75 times more efficiently than Y12M and is about twice as fast as MA28. For the ring modulator and EMEP problem, the four solvers show roughly the same statistics. They slightly depend on m, since the order in which pivots are selected within diagonal blocks of $I_d - h_n d_i J_n$ may differ. For the pollution problem all integration statistics were identical to the LINPACK statistics.

Secondly, we investigate how the parallel performance of PSODE depends on the solver and on m. We implemented the codes on the Cray C98/4256 at SARA. Since the integration statistics were roughly the same as in the Tables 6.2, 6.3 and 6.4, we only listed the speed-up factors of the runs on 4 processors compared to the runs in one-processor mode. The Cray C98/4256 is a shared memory computer with 4 processors. 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 [Cra94b]

TABLE 6.4: *Results on EMEP problem.*

	m	CPU	scd	steps	J-eval	LU-dec	f-eval
LINPACK	1	107.27	3.75	594	243	716	4422
	2	404.32	3.84	614	273	722	4516
	3	861.06	3.77	567	260	676	4251
	10	9951.05	3.91	621	260	719	4507
MA28	1	80.36	3.81	645	267	772	4723
	2	142.15	3.56	549	251	660	4132
	3	228.44	3.73	635	250	749	4592
	10	690.34	3.81	530	238	626	3973
Y12M	1	56.66	3.95	574	250	685	4992
	2	116.46	3.75	618	262	738	4553
	3	180.10	3.97	654	262	773	4705
	10	581.59	3.63	589	260	689	4369
SPS	1	33.09	3.11	606	264	722	4499
	2	57.78	3.64	581	246	681	4270
	3	89.77	3.55	605	256	699	4390
	10	352.61	3.89	509	229	643	3924

TABLE 6.5: *Speed-up factors for the three test problems.*

		ring modulator		pollution problem		EMEP problem	
		predicted speed-up	optimal speed-up	predicted speed-up	optimal speed-up	predicted speed-up	optimal speed-up
LINPACK	1	2.2	2.7	2.2	2.5	3.4	3.5
	2	2.6	3.0	2.7	3.0	3.6	3.8
	3	3.0	3.2	3.1	3.3	3.6	3.9
	10	3.6	3.8	3.5	3.8	3.5	3.9
Y12M	1	3.1	3.4	2.8	3.4	3.8	3.8
	2	3.3	3.5	2.8	3.5	3.8	3.8
	3	3.4	3.6	3.5	3.6	3.8	3.8
	10	3.5	3.7	3.6	3.7	3.8	3.9
SPS	1	2.4	2.8	2.4	2.7	3.6	3.7
	2	2.6	2.9	2.7	2.9	3.7	3.8
	3	2.8	3.1	2.5	3.0	3.8	3.8
	10	3.3	3.4	3.0	3.3	3.7	3.8

to predict the speed-up factors on 4 processors. Table 6.5 gives these results. 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 6.5.

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 [Cra94a] for the specification of these settings. We did not include results for MA28, since for a parallel implementation of this code, one would have to get rid of the common blocks. Table 6.5 confirms the expectation that the speed-up factors grow for increasing problem sizes. The predicted speed-up factors do not always increase monotonically with m. This can be explained by the fact that the results of ATExpert are based on a varying number of processors. For PSODE with LINPACK the optimized routines `sgefa` and `sgesl` of the Cray library were used. The relatively fast performance of the resulting code leads to smaller speed-up factors. The speed-up of SPS is somewhat withdrawn with respect to Y12M for large m. We explain this by the smaller amount of computations that have to be done in SPS before communicating.

6.6 Conclusions

In this chapter we tested the direct sparse matrix solvers MA28, Y12M and one special purpose solver in the parallel ODE solver PSODE. If the number of nonzeros in the Jacobian of the derivative function is less than about 20–25% of the total number of entries, then it begins to pay off to use a sparse matrix solver. The costs can be further reduced by a factor varying from 1.75 to 2 by using a special purpose solver based on Gaussian Elimination and diagonal Markowitz pivoting without stability check. Up to a certain extent we can explain theoretically why numerically solving stiff ODEs with the use of a Newton-type process for the solution of the systems of non-linear equations, leads to linear systems that can be solved without stability pivoting.

Experiments on a Cray C98/4256 show speed-up factors of PSODE on 4 processors in the region 2.2–3.8, depending on the problem size, but not much on the (sparse) matrix solver.

Chapter 7

Parallel Linear System Solvers

Abstract If the nonlinear systems arising in implicit Runge–Kutta methods like the Radau IIA methods are iterated by (modified) Newton, then we have to solve linear systems whose matrix of coefficients is of the form $I - A \otimes hJ$ with A the Runge–Kutta matrix and J an approximation to the Jacobian of the right-hand side function of the system of differential equations. For larger systems of differential equations, the solution of these linear systems by a direct linear solver is very costly, mainly because of the LU-decompositions. We try to reduce these costs by solving the linear systems by a second (inner) iteration process. This inner iteration process is such that each inner iteration again requires the solution of a linear system. However, the matrix of coefficients in these new linear systems is of the form $I - B \otimes hJ$ where B is similar to a diagonal matrix with positive diagonal entries. Hence, after performing a similarity transformation, the linear systems are decoupled into s subsystems, so that the costs of the LU-decomposition are reduced to the costs of s LU-decompositions of dimension d. Since these LU-decompositions can be computed in parallel, the effective LU-costs on a parallel computer system are reduced by a factor s^3. It will be shown that matrices B can be constructed such that the inner iterations converge whenever A and J have their eigenvalues in the positive and nonpositive half-plane, respectively. The theoretical results will be illustrated by a few numerical examples. A parallel implementation on the four-processor Cray-C98/4256 shows a speed-up ranging from at least 2.4 until at least 3.1 with respect to RADAU5 applied in one-processor mode.

7.1 Introduction

Suppose that we integrate the IVP

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}} \tag{7.1}$$

by an implicit step-by-step method. In general, this requires in each step the solution of a nonlinear system of the form

$$R(Y_n) = 0, \quad R(Y) := Y - h(A \otimes I)F(Y) - W_{n-1}, \tag{7.2}$$

where A denotes an $s \times s$ matrix (assumed to be nondefective), I is the $d \times d$ identity matrix, W_{n-1} contains information from preceding steps, h is the stepsize $t_n - t_{n-1}$, and $\otimes$ denotes the Kronecker product. It will always be assumed that A is nondefective and has its eigenvalues in the positive half-plane. The s components Y_{ni} of the sd-dimensional solution vector Y_n represent s numerical approximations to the s exact solution vectors $y(t_{n-1} + c_i h)$; here, $c = (c_i)$ denotes the abscissa vector whose components c_i are assumed distinct. Furthermore, for any vector $Y_n = (Y_{ni})$, $F(Y_n)$ contains the derivative values

$f(Y_{ni})$. In the following, we shall use the notation I for any identity matrix. However, its order will always be clear from the context. The solution Y_n of (7.2) will be called the stage vector, y_n the step point value, s the number of stages, and A the Runge–Kutta matrix.

Usually, the nonlinear system (7.2) is solved by modified Newton iteration. This leads to linear systems whose matrix of coefficients is of the form $I - A \otimes hJ$ with J an approximation to the Jacobian of the right-hand side function f. The solution of these systems may be extremely costly. For example, if a direct solver is used, then in general the LU-decomposition requires $(2/3)s^3d^3$ arithmetic operations which is considerable, even for moderate values of d (say $d \approx 10$). Moreover, there is only a limited intrinsic parallelism in building the LU-decomposition of the matrix $I - A \otimes hJ$.

7.1.1 Reduction of computational costs

We briefly survey various approaches to reduce the computational costs associated with the solution of the Newton systems using parallel computer systems. Firstly, one may look for special methods in which A is a triangular matrix with positive diagonal entries like the DIRK type methods. Then, confining our considerations to the costs of the LU-decomposition, we see that the effective LU-costs on s processors reduce to $(2/3)d^3$ operations, a factor s^3 less than those needed for the Newton process. However, these DIRK type methods also have disadvantages. In the case of one-step DIRKs available in the literature, the step point order is at most 4 and they have a relatively low stage order which may be a disadvantage in certain classes of stiff IVPs. Higher step point orders and stage orders can be obtained in the class of multistep RK methods (cf. Burrage and Chipman [BC89]), but they have the disadvantage of quite large abscissae values c_i (much larger than 1).

More sophisticated than the DIRK methods are methods characterized by matrices A with only positive eigenvalues such as the one-step RK methods of Nørsett [Nør76], Burrage [Bur78] and Orel [Ore93]. By performing a similarity transformation (or Butcher transformation [But76]), the linear systems can be decoupled into s subsystems of dimension d. Again, the effective LU-costs reduce by a factor s^3, and moreover, the stage order and step point order are much higher than for DIRK methods. However, a possible disadvantage of these methods is the lack of superconvergence at the step points.

Finally, one may choose the classical RK methods possessing both a high stage order and a high step point order, but also one or more complex eigenvalues. Again, applying a similarity transformation, the Newton system is transformed to block-diagonal form with (real) diagonal blocks, each block corresponding to an eigenvalue of A. If an eigenvalue is real, then the associated diagonal block is of order d, otherwise it has order $2d$. The LU-costs of these blocks are reduced to $(2/3)d^3$ and $(16/3)d^3$ operations, so that effectively the LU-costs are $(16/3)d^3$ operations, irrespective the value of s (the code RADAU5 of Hairer and Wanner [HW91] uses such a transformation).

7.1.2 Iterative solution of the linear systems

Instead of using direct solution methods, one may also look for iterative linear solvers, such as GMRES or preconditioned GMRES (see e.g. Burrage [Bur95] where further references are given).

In this chapter, we shall follow an approach that is a mixture of an iterative and a direct approach. It allows A to have complex eigenvalues (in the positive half-plane), so that the superconvergent RK methods like the Radau IIA methods are included. The linear systems arising in the modified Newton method are solved by an iterative method (the inner iteration process), which needs itself LU-decompositions of matrices, but these matrices are only of dimension d. In fact, the linear systems to be solved have a matrix of coefficients of the form $I - B \otimes hJ$ where B is similar to a diagonal matrix with positive diagonal entries. Hence, after performing a similarity transformation, the effective LU-costs are $(2/3)d^3$ operations like the methods of Burrage and Orel. We shall refer to this inner iteration process by PILSRK (Parallel Iterative Linear System solver for RK methods). The combination of the modified Newton and the PILSRK method will be called the Newton–PILSRK method.

There are several options for choosing the matrix B. The most simple approach chooses $B = D$ where D is a diagonal matrix (with positive entries), so that the sd-dimensional system can directly be split into s uncoupled subsystems of dimension d which can be solved concurrently. In fact, we can employ the same matrices D as used in the Parallel Diagonal-implicitly Iterated RK methods (PDIRK methods) analyzed in [HS91]. The PDIRK method is also an iterative method, but unlike the PILSRK method it is a nonlinear system solver and directly iterates on the nonlinear system (7.2). Using results derived by Lioen [Lio96] for PDIRK matrices, it can be shown that for the first eight Radau IIA correctors, the PILSRK methods are A-convergent, that is, it converges if J has its eigenvalues in the nonpositive half-plane. Furthermore, these PDIRK matrices have the property that the stiff components are removed from the iteration error within s iterations. However, a disadvantage of the PDIRK matrices is the poor convergence (or even divergence) of the PILSRK method in the first few iterations which is worse as the number of stages of the underlying RK corrector increases. Such a convergence behavior is highly undesirable if we want to apply step-parallel iteration, where the iteration process is already started at the next step point t_{n+1} before the iterates at t_n have converged. A poor initial convergence implies that no accurate predictor value is available for starting the iteration process at t_{n+1}.

A substantial improvement in the initial phase of the convergence of the PILSRK method is obtained by employing the matrices L used in the Parallel Triangular-implicitly Iterated RK methods (PTIRK methods) constructed in Chapter 4 (like the PDIRK methods, the PTIRK methods are nonlinear system solvers). The PTIRK matrices L are defined by the lower triangular factor of the Crout decomposition LU of the RK matrix A. By virtue of results obtained in Chapter 5 it can be shown that for all RK correctors that are based on collocation with positive, distinct abscissae, the matrix L has positive diagonal entries and that the PILSRK method is A-convergent. Furthermore, like

the PDIRK matrices, the PTIRK matrices have the property that the stiff components are removed from the iteration error within s iterations. After performing a similarity transformation, the effective LU-costs are reduced by a factor s^3. A preliminary parallel implementation of the Newton–PILSRK method based on the one-step 4-stage Radau IIA formula and using the PTIRK matrix showed on the four-processor Cray-C98/4256 speed-up factors ranging from at least 2.4 until at least 3.1 with respect to RADAU5 in one-processor mode (cf. Chapter 4).

7.1.3 Outline of the chapter

The aim of this chapter is to find matrices B that are still more effective than the PTIRK matrices L. Our starting point is the representation $B = QTQ^{-1}$ with T a lower triangular matrix with positive diagonal entries and with Q a nonsingular transformation matrix such that $Q^{-1}AQ$ is lower block- triangular. It will be shown that matrices T and Q exist such that:

1. B is nondefective and has positive eigenvalues,
2. the PILSRK method is A-convergent whenever A has its eigenvalues in the positive half-plane,
3. the stiff components are removed from the iteration error in the second iteration,
4. the spectral radius of the iteration-error-amplification matrix is minimized in the left-hand half-plane.

The difficult part is the construction of matrices Q such that the iteration-error-amplification matrix has a sufficiently small norm. In this chapter, we construct transformation matrices so that $Q^{-1}AQ$ is block-diagonal. For the 4-stage and 8-stage Radau IIA correctors, matrices Q will be constructed such that the Euclidean norm of powers of the iteration-error-amplification matrix are satisfactorily small.

As soon as T and Q, and hence B, are obtained, we can compute the diagonalizing similarity transformation, to obtain a highly parallel linear system solver.

In this chapter, we have restricted our analysis of the Newton–PILSRK method to the case where (7.2) represents the class of one-step Radau IIA methods, that is, A is the Radau IIA matrix and $W_{n-1} := (E \otimes I)Y_{n-1}$ with $E = (0, \ldots, 0, \mathbb{1})$, $\mathbb{1}$ being an s-dimensional vector with unit entries. These methods are of particular interest because of their high step point order $p = 2s - 1$ and high stage order $q = s$, their stiff accuracy and their excellent stability properties. The Newton–PILSRK methods were applied to a few problems taken from the literature. The results show a considerable improvement of the convergence in the first few outer iterations. Recalling that a parallel implementation of the Newton–PILSRK method using the PTIRK matrices already shows a speed-up factor of at least 2.4 with respect to RADAU5, we expect that using the new matrices $B = QTQ^{-1}$ will yield a further speed-up. The parallel implementation of the new methods will be subject of future research.

Finally, we remark that it may well be that the class of multistep RK methods of Radau type (cf. Hairer and Wanner [HW91, p. 293]) is a better choice for the corrector equation (7.2) than the one-step Radau methods. For non-stiff IVPs, Burrage and Suhartanto [BS97] have investigated the use of parallel iteration methods for such correctors and they report promising results. This indicates that applying the PILSRK approach of this chapter to the Newton systems arising in multistep Radau methods may lead to quite effective parallel IVP methods. Chapter 8 will deal with this class of methods.

7.2 The parallel iterative linear system solver

Consider the modified Newton iteration scheme for solving the corrector equation (7.2):

$$(I - A \otimes hJ)(Y^{(j)} - Y^{(j-1)}) = -R(Y^{(j-1)}), \qquad j = 1, 2, \ldots, m, \tag{7.3}$$

where $J = \partial f/\partial y$ is evaluated at t_{n-1}, $Y^{(0)}$ is the initial iterate to be provided by some predictor formula, and where $Y^{(m)}$ is adopted as the solution Y_n of the corrector equation (7.2). Each iteration with (7.3) requires the solution of an sd-dimensional linear system for the Newton correction $Y^{(j)} - Y^{(j-1)}$. As already observed, direct solution of this Newton system can be extremely costly and transformation to block-diagonal form reduces computational costs considerably. In order to achieve a still greater reduction of the computational complexity we follow an alternative approach by applying an iterative linear solver to the Newton systems in (7.3). This solver again requires the solution of linear systems, but these systems are only of dimension d. It is tuned to the RK structure of the systems in (7.3) and possesses a lot of intrinsic parallelism. This Parallel Iterative Linear System solver for RK methods (PILSRK method) is defined by

$$\begin{aligned} &(I - B \otimes hJ)(Y^{(j,\nu)} - Y^{(j,\nu-1)}) = -(I - A \otimes hJ)Y^{(j,\nu-1)} + C^{(j-1)}, \\ &C^{(j-1)} := (I - A \otimes hJ)Y^{(j-1)} - R(Y^{(j-1)}), \qquad \nu = 1, 2, \ldots, r, \end{aligned} \tag{7.4}$$

where $Y^{(j,0)} = Y^{(j-1,r)}$ and where $Y^{(m,r)}$ is accepted as the solution Y_n of the corrector equation (7.2). The matrix B is assumed to be nondefective and to have positive eigenvalues. Note that $C^{(j-1)}$ does not depend on ν, so that the application of the inner iteration process requires only one evaluation of the function R. The processes (7.3) and (7.4) may be considered as the *outer* and *inner* iteration processes.

In order to construct a suitable matrix B, we observe that the condition on the spectrum of B implies that we can write $B = QTQ^{-1}$ with Q an arbitrary real, nonsingular matrix and T a lower triangular matrix with positive diagonal entries. Hence, by performing the transformation

$$Y^{(j,\nu)} = (Q \otimes I)\widetilde{Y}^{(j,\nu)},$$

we obtain

$$(I - T \otimes hJ)(\widetilde{Y}^{(j,\nu)} - \widetilde{Y}^{(j,\nu-1)}) = -(I - \widetilde{A} \otimes hJ)\widetilde{Y}^{(j,\nu-1)} + (Q^{-1} \otimes I)C^{(j-1)}, \tag{7.5}$$

where $\nu = 1, 2, \ldots, r$, $\widetilde{A} = Q^{-1}AQ$ and $\widetilde{Y}^{(j,0)} = (Q^{-1} \otimes I)Y^{(j-1)}$. If for a given j, the transformed inner iterates $\widetilde{Y}^{(j,\nu)}$ converge to a vector $\widetilde{Y}^{(j,\infty)}$, then the Newton iterate defined by (7.3) can be obtained from $Y^{(j)} = (Q \otimes I)\widetilde{Y}^{(j,\infty)}$. Given the matrix A, the PILSRK method (7.5) is completely defined by the matrix pair (T,Q) and will be denoted by PILSRK(T,Q). The representation (7.5) will be the starting point for the construction of the matrix B.

Before discussing the computational costs of the implementation of the Newton–PILSRK method {(7.3),(7.5)}, we should specify the matrix B. This will be subject of §7.3. Details on the computational complexity can be found in §7.4.2.

REMARK In the first Newton iterations, it seems a waste to perform many inner iterations with the PILSRK method, because there is no point in computing a very accurate approximation to $Y^{(j)}$, as long as $Y^{(j)}$ is itself a poor approximation to Y_n. Likewise, in later outer iterations, we expect that only a few inner iterations suffice to solve $Y^{(j)}$ from (7.3). In the extreme case, only one inner iteration is performed in each outer iteration. In such an iteration strategy, the Newton–PILSRK iteration method {(7.3),(7.4)} simplifies to

$$(I - B \otimes hJ)(Y^{(j)} - Y^{(j-1)}) = -R(Y^{(j-1)}), \qquad j = 1, 2, \ldots, m. \tag{7.6}$$

However, this process may converge very slowly in the first few outer iterations, and it is recommended either to use highly accurate predictor formulas for $Y^{(0)}$, or to introduce a dynamic iteration strategy so that when necessary, sufficiently many inner iterations in the first few outer iterations are performed. Notice also that the iterative method obtained from (7.3) by using a splitting of A into B and $A - B$ is identical with the iteration method (7.6). ◇

7.3 Construction of the matrix B

Given the matrix A, the PILSRK method (7.4) is completely determined by the matrix $B = QTQ^{-1}$. In the construction of B, the region of convergence and the averaged amplification factors for the iteration errors play a central role.

7.3.1 Convergence region of the PILSRK method

In order to analyze the region of convergence for the PILSRK method, we consider the error recursion

$$Y^{(j,\nu)} - Y^{(j)} = M(Y^{(j,\nu-1)} - Y^{(j)}), \quad M := (I - B \otimes hJ)^{-1}((A - B) \otimes hJ). \tag{7.7}$$

We have convergence if the powers M^ν of the amplification matrix M tend to zero as $\nu \to \infty$, that is, if the spectral radius $\rho(M)$ of M is less than 1. The eigenvalues of M are given by the eigenvalues of the matrix

$$Z(z) := z(I - zB)^{-1}(A - B), \qquad z := h\lambda, \tag{7.8}$$

where λ runs through the eigenvalues of J. We call $\Gamma := \{z : \rho(Z(z)) < 1\}$ the region of convergence of the PILSRK method. Thus, the method converges if the eigenvalues of hJ lie in Γ. If Γ contains the whole nonpositive half-plane, then the method will be called *A-convergent.*

We shall call $Z(z)$ the amplification matrix at the point z and $\rho(Z(z))$ the *(asymptotic) amplification factor* at z. The maximum of $\rho(Z(z))$ in the left-hand half-plane $\mathrm{Re}(z) \leq 0$ will be denoted by ρ.

In [HS91] and Chapter 4 where the PDIRK and PTIRK methods are analyzed, it turned out that strong damping of the stiff error components, that is, small amplification factors for error components corresponding to eigenvectors of J with eigenvalues λ of large magnitude, is crucial for a fast overall convergence. This leads us to require the matrix B to be such that $\rho(Z(\infty)) = \rho(I - B^{-1}A)$ vanishes. If we succeed in finding such matrices B, then $Z^s(\infty) = 0$, so that within s iterations, the components corresponding to $|\lambda| = \infty$ are removed from the iteration error (this can be verified by considering the Schur decomposition of $Z^s(\infty)$).

As an example, let $Q = I$ and let T be a diagonal matrix D, so that $B = D$. Lioen [Lio96] showed that for the s-stage Radau IIA correctors with $s \leq 8$, it is possible to construct diagonal matrices D satisfying $\rho(I - D^{-1}A) = 0$ such that the generated PILSRK(D,I) method is A-convergent. These matrices are also used in the PDIRK methods studied in [HS91], and will therefore be called PDIRK matrices.

The next theorem defines a family of PILSRK(T,Q) methods automatically satisfying the condition $\rho(I - B^{-1}A) = 0$.

THEOREM 7.1 *Let Q be an arbitrary, nonsingular matrix and let $B = QTQ^{-1}$, where T is the lower triangular factor in the Crout-decomposition of $\widetilde{A} := Q^{-1}AQ$. Then, the asymptotic amplification factor vanishes at infinity.*

PROOF Let TU represent the Crout-decomposition of $\widetilde{A}$. Then

$$Q^{-1}Z(\infty)Q = I - Q^{-1}B^{-1}AQ = I - T^{-1}\widetilde{A} = I - U$$

is strictly upper triangular. Hence, $\rho(Q^{-1}Z(\infty)Q) = \rho(Z(\infty)) = 0$. □

The matrix B in the PILSRK methods characterized by this theorem does not necessarily have positive eigenvalues and hence, does not automatically generate A-convergent methods. This requires special transformation matrices Q. Let us again consider the case where $Q = I$. Then, B equals the lower triangular factor in the Crout-decomposition of A, that is, B equals the PTIRK matrix L derived in Chapter 4. In Chapter 5 it is proved that the PTIRK matrix L possesses positive diagonal entries for all collocation-based RK correctors with positive, distinct abscissas, so that B has positive eigenvalues as required. Furthermore, numerical computations in Chapter 4 show the A-convergence for a large number of RK correctors based on Gaussian quadrature formulas.

The aim of this chapter is to derive A-convergent methods with $\rho(I - B^{-1}A) = 0$ for more general pairs (T,Q) than the PTIRK pair (L,I), and to find pairs (T,Q) such that

we can a priori prove both the positiveness of the eigenvalues of B and the A-convergence of the generated iteration method.

Let us choose Q such that $\widetilde{A} := Q^{-1}AQ = (\widetilde{A}_{kl})$ is a (real) $\sigma \times \sigma$ lower block-triangular matrix, of which the diagonal blocks $\widetilde{A}_{kk}$ are either one-by-one or two-by-two matrices. If ξ_k is a real eigenvalue of A, then $\widetilde{A}_{kk} = \xi_k$, and if $\xi_k \pm i\eta_k$ is a complex eigenvalue pair of A, then

$$\widetilde{A}_{kk} = \begin{bmatrix} a_k & b_k \\ c_k & 2\xi_k - a_k \end{bmatrix},$$

$$b_k = -c_k^{-1}(a_k^2 - 2\xi_k a_k + \alpha_k^2), \quad c_k \neq 0, \quad \alpha_k := \sqrt{\xi_k^2 + \eta_k^2}, \tag{7.9}$$

where a_k and c_k are free parameters. In the following, K will denote the set of integers with the property that $\eta_k \neq 0$ whenever $k \in K$.

A natural choice for T now is

$$T := \begin{bmatrix} T_{11} & 0 & 0 & 0 & \cdots \\ \widetilde{A}_{21} & T_{22} & 0 & 0 & \cdots \\ \widetilde{A}_{31} & \widetilde{A}_{32} & T_{33} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

$$T_{kk} := \begin{cases} \begin{bmatrix} u_k & 0 \\ v_k & w_k \end{bmatrix}, & \text{if } k \in K, \\ \xi_k & \text{otherwise}, \end{cases} \tag{7.10}$$

where u_k, v_k and w_k are free parameters with u_k and w_k assumed to be positive.

THEOREM 7.2 *Let A have its eigenvalues in the positive half-plane, let $\widetilde{A} := Q^{-1}AQ = (\widetilde{A}_{kl})$ be lower block-triangular, let the diagonal blocks be defined by (7.9) and let $T = T(\gamma)$ be defined by (7.10) with*

$$u_k = \gamma\alpha_k, \quad v_k = c_k\alpha_k \frac{(\gamma^2 - 1)a_k - 2\gamma^2\xi_k + 2\gamma\alpha_k}{\gamma(a_k^2 - 2\xi_k a_k + \alpha_k^2)}, \quad w_k = \frac{\alpha_k}{\gamma}, \tag{7.11}$$

where γ is a positive parameter. Then, for all a_k and c_k the following assertions hold:

1. *The asymptotic amplification factor vanishes at infinity.*
2. *B has positive eigenvalues and if $\gamma \neq 1$ it is nondefective.*
3. *The PILSRK($T(\gamma)$,Q) method is A-convergent with*

$$\rho = \max\{|1 - 2\gamma(\gamma^2 + 1)^{-1}\xi_k\alpha_k^{-1}| : k \in K\}.$$

PROOF Let

$$\widetilde{Z}(z) := Q^{-1}Z(z)Q = z(I - zT)^{-1}(\widetilde{A} - T), \qquad z := h\lambda. \tag{7.12}$$

If T is of the form (7.10), then the value of $\rho(Z(z)) = \rho(\widetilde{Z}(z))$ equals the maximum of the spectral radius $\rho(\widetilde{Z}_{kk}(z))$ of the diagonal blocks

$$\widetilde{Z}_{kk} := z(I - zT_{kk})^{-1}(\widetilde{A}kk - T_{kk}) \tag{7.13}$$

of $\widetilde{Z}$. Here, $\widetilde{Z}_{kk}$ vanishes if the underlying eigenvalue of A is real. Hence, in order to achieve $\rho(Z(\infty)) = 0$, we choose the T_{kk} with $k \in K$ such that the spectral radius of the corresponding diagonal blocks $Z_{kk}(z)$ vanishes at infinity.

We derive from (7.9) and (7.13) that the eigenvalues ζ_k of $\widetilde{Z}_{kk}$ satisfy the characteristic equation

$$\det \begin{bmatrix} (a_k - u_k)z - \zeta_k(1 - zu_k) & b_k z \\ (c_k - v_k)z + \zeta_k v_k z & (2\xi_k - a_k - w_k)z - \zeta_k(1 - zw_k) \end{bmatrix} = 0. \tag{7.14}$$

It is easily verified that $\zeta_k = \zeta_k(z)$ vanishes at infinity if u_k, v_k and w_k are defined according to (7.11). Hence, $\rho(\widetilde{Z}_{kk}(z))$ vanishes at infinity which proves the first part of the theorem.

Since the eigenvalues of B are given by $\{u_k, w_k\}$ for $k \in K$ and by ξ_k for $k \notin K$, and because we assumed $\gamma > 0$, (7.11) also implies that B has positive eigenvalues and if $\gamma \neq 1$ it is nondefective, proving the second part.

The characteristic equation (7.14) is solved by

$$\zeta_k = 0, \qquad \zeta_k = \frac{(2\xi_k - u_k - w_k)z}{(1 - zu_k)(1 - zw_k)}, \tag{7.15}$$

so that $\rho(\widetilde{Z}(z))$ equals the maximum of the values $|\zeta_k(z)|$. Since $\zeta_k(z)$ is regular in left-hand half-plane (provided that u_k and w_k are positive), its maximum in the left-hand half-plane $\mathrm{Re}(z) \leq 0$, to be denoted by ρ_k, is assumed on the imaginary axis. It is easily verified that

$$\rho(\widetilde{Z}_{kk}(\mathrm{i}y)) = |\zeta_k(\mathrm{i}y)| = \frac{|2\xi_k - u_k - w_k||y|}{\sqrt{(1 + u_k^2 y^2)(1 + w_k^2 y^2)}} \tag{7.16}$$

assumes an absolute maximum at $y = y_0 := (u_k w_k)^{-1/2}$ and that the maximum value ρ_k of $\rho(Z_{kk}(\mathrm{i}y))$ is given by

$$\rho_k = |1 - 2\xi_k(u_k + w_k)^{-1}| = |1 - 2\gamma(\gamma^2 + 1)^{-1}\xi_k \alpha_k^{-1}|,$$

which is less than 1 whenever $\gamma\xi_k > 0$. This proves the last part of the theorem. □

The asymptotic amplification factor ρ is minimized for $\gamma = 1$ and assumes the minimal value $\rho = \max\{1 - \xi_k\alpha_k^{-1} : k \in K\}$. However, then the matrices T_{kk} are defective

(because $u_k = w_k$). Hence, T cannot be diagonalized, and although the effective LU-costs are still reduced by a factor s, the Newton–PILSRK($T(1)$,Q) method should be considered as a σ-processor method, rather than an s-processor method. Fortunately, the asymptotic amplification factor varies slowly with γ, so that we can remove the defectness of T at the cost of a slight increase of ρ. For example, for the method defined by {(7.10),(7.11)} we find for $\gamma = 7/8$,

$$\rho = \max\{1 - \tfrac{112}{113}\xi_k\alpha_k^{-1} : k \in K\}, \tag{7.17}$$

which is only slightly larger than the minimal value. For a detailed discussion of the computational complexity of an implementation of the Newton–PILSRK($T(\gamma)$,Q) method, we refer to §7.4.2.

REMARK 1 When faced with the problem of choosing a matrix T such that the eigenvalues of the matrix $Z(z)$ are of small magnitude, it is tempting to minimize the magnitude of the matrix factor $\widetilde{A} - T$ occurring in the matrix $\widetilde{Z}(z)$ defined by (7.12). Since

$$\begin{aligned} \widetilde{A} - T &= \operatorname{diag}\{\widetilde{A}_{11} - T_{11}, \ldots, \widetilde{A}_{\sigma\sigma} - T_{\sigma\sigma}\}, \\ \widetilde{A}_{kk} - T_{kk} &= \begin{bmatrix} a_k - u_k & b_k \\ c_k - v_k & 2\xi_k - a_k - w_k \end{bmatrix} \end{aligned}$$

and because for given a_k, the magnitude of the entry $b_k = -c_k^{-1}(a_k^2 - 2\xi_k a_k + \alpha_k^2)$ can be made as small as we want, we are led to zero the other three entries of $\widetilde{A}_{kk} - T_{kk}$ by setting $u_k = a_k$, $v_k = c_k$ and $w_k = 2\xi_k - a_k$. This still leaves a_k as a free parameter which can be used to minimize b_k for given c_k, to obtain $a_k = \xi_k$ and $b_k = -\eta_k^2 c_k^{-1}$. However, substitution of the parameters u_k, v_k, w_k, a_k and b_k into the characteristic equation (7.14) reveals that the nonzero eigenvalue is given by $\zeta_k = (\xi_k^2 - \alpha_k^2)z^2(1 - z\xi_k)^{-2}$, which assumes the extreme value $-(\eta_k\xi_k^{-1})^2$ at infinity. Thus, we have no A-convergence when A has eigenvalues whose imaginary part exceeds its real part. Since many RK methods based on Gaussian quadrature do have imaginary parts that exceed the real parts, the approach of minimizing the magnitude of $\widetilde{A} - T$ is the wrong way to go. ◇

REMARK 2 The family of matrices T defined by {(7.10),(7.11)} contains the special case where T is defined by the lower triangular factor in the Crout-decomposition of $\widetilde{A} := Q^{-1}AQ$ (see Theorem 7.1):

$$T_{kk} := \begin{cases} \begin{bmatrix} a_k & 0 \\ c_k & \alpha_k^2 a_k^{-1} \end{bmatrix} & \text{if } k \in K, \\ \xi_k & \text{otherwise.} \end{cases} \tag{7.18}$$

This expression is also obtained from {(7.10),(7.11)} by setting $\gamma = a_k\alpha_k^{-1}$. ◇

We conclude this section with listing values of ρ_k for a few Radau IIA correctors and for the iteration strategy PILSRK($T(7/8)$,Q) defined by Theorem 7.2. In addition, we list the values of ρ for PILSRK(D,I) with the PDIRK matrix D and for PILSRK(L,I) with the PTIRK matrix L. The figures in Table 7.1 show that on the basis of the asymptotic amplification factors, the PILSRK($T(7/8)$,Q) approach is superior to PILSRK(D,I) and PILSRK(L,I).

TABLE 7.1: *Values of ρ_k for Radau IIA methods.*

Iteration	k	$s=2$	$s=3$	$s=4$	$s=6$	$s=8$
PILSRK(D,I)	-	0.26	0.40	0.52	0.72	0.90
PILSRK(L,I)	-	0.18	0.37	0.51	0.70	0.86
PILSRK($T(7/8)$,Q)	1	0.19	0.35	0.45	0.57	0.64
	2			0.06	0.21	0.33
	3				0.03	0.12
	4					0.02

7.3.2 Averaged amplification factors

Because the matrix M in (7.7) is not expected to be a normal matrix, the asymptotic amplification factor ρ discussed in the preceding section only gives an indication of the speed of convergence after a quite large number of iterations and does not give insight into the convergence behavior in the initial phase of the iteration process. In fact, for large ν we have the estimate $\|M^\nu\| \leq \kappa(S)(\rho(M))^\nu$, where S represents the eigensystem of M, $\kappa(S) := \|S\|\,\|S^{-1}\|$ is the condition number of S, and where we assumed that M has eigenvalues of multiplicity 1 (cf. Varga [Var62]). In order to analyze the convergence rate in the first few iterations, one may resort to the pseudo-eigenvalue analysis of Trefethen (see e.g. [RT92]). Alternatively, we may resort to a well-known theorem of Von Neumann. We shall follow the latter approach.

Let the logarithmic matrix norm $\mu[S]$ associated with the Euclidean norm be defined by $\mu[S] = (1/2)\lambda_{\max}(S + S^{\mathrm{H}})$, where S^{H} is the complex transposed of S and $\lambda_{\max}(\cdot)$ denotes the algebraically largest eigenvalue. Then, we have:

THEOREM 7.3 *If $\mu[J] \leq 0$, then $\|M^\nu\| \leq \max\{\|Z^\nu(z)\| : \mathrm{Re}(z) \leq 0\}$.*

PROOF The proof is based on a generalization of a theorem of Von Neumann. Von Neumann's theorem states that, given a matrix J and a rational function R of z which has a bounded maximum norm $\|R\|_\infty$ in the left-hand half-plane, then $\|R(J)\| \leq \|R\|_\infty$, provided that $\mu[J] \leq 0$ (see e.g. [HW91, p. 179]). A matrix-valued version of Von Neumann's theorem, applying to the case where $R(z)$ is a matrix with entries that are rational functions of z, was proved by Nevanlinna [Nev85] (see also [HW91, p. 356]).

Since M^ν can be considered as a matrix-valued function of J (see (7.7)), we apply the matrix-valued version of Von Neumann's theorem with $R(z) := M^\nu(z)$, where

$$M^\nu(z) = [(I - B \otimes zI)^{-1}((A - B) \otimes zI)]^\nu = Z^\nu(z) \otimes I, \; z = h\lambda. \tag{7.19}$$

This leads to the assertion of the theorem. □

This theorem motivates us to define the local averaged amplification factor at the point $z = h\lambda$ and the global averaged amplification factor by

$$\rho^{(\nu)}(z) := \sqrt[\nu]{\|Z^\nu(z)\|}, \qquad \rho^{(\nu)} := \max\{\rho^{(\nu)}(z) : \mathrm{Re}(z) \leq 0\}. \tag{7.20}$$

Note that $\rho^{(\nu)}(z)$ approximates the asymptotic amplification factor $\rho(Z(z))$ as $\nu \to \infty$. Since in the left-hand half-plane, $\rho^{(\nu)}(z)$ assumes its maximum on the imaginary axis, we may restrict our considerations to the imaginary axis, so that $\rho^{(\nu)} := \max\{\rho^{(\nu)}(\mathrm{i}y) : y \geq 0\}$.

Theorem 7.3 indicates that we may expect faster convergence as $\rho^{(\nu)}$ is smaller. However, for small numbers of iterations (say $\nu \leq 5$), $\rho^{(\nu)}$ will give a rather conservative estimate of the speed of convergence, because in some sense it is a 'worst case' estimate. In order to get insight into the amplification of *individual* error components, one may use the *local* amplification factor $\rho^{(\nu)}(z)$. Let us consider error components of the form $a \otimes v$, where a is an s-dimensional vector and v is an eigenvector of J with eigenvalue l. By observing that $M^\nu(a \otimes v) = (Z^\nu(h\lambda) \otimes I)(a \otimes v)$, it follows that $\rho^{(\nu)}(h\lambda)$ characterizes the averaged convergence of the error component corresponding with $h\lambda$ and that only for larger values of ν, when the error component with maximal $\rho^{(\nu)}(h\lambda)$ has become dominant, $\rho^{(\nu)}$ yields a quantitative estimate of the averaged convergence rate. In the first few iterations, when all error components play their part, the L_2 norm of the local amplification factor $\rho^{(\nu)}(z)$ provides more realistic estimates than the L_∞ norm. This suggests to define a second global amplification factor:

$$\sigma^{(\nu)} := \left(\int_0^\infty [\rho^{(\nu)}(\mathrm{i}y)]^2 \mathrm{d}y \right)^{1/2}. \tag{7.21}$$

We did not succeed in finding an approach which really minimizes $\rho^{(\nu)}$. However, by considering the estimate

$$\|Z^\nu(z)\| = \|Q\widetilde{Z}^\nu(z)Q^{-1}\| \leq \kappa(Q)\|\widetilde{Z}^\nu(z)\|, \tag{7.22}$$

we see that $\rho^{(\nu)}(z) \leq (\kappa(Q)\|\widetilde{Z}^\nu(z)\|)^{1/\nu}$, which suggests the separate minimization of the factors $\kappa(Q)$ and $\|\widetilde{Z}^\nu(z)\|$. We distinguish two approaches. In the first approach, we choose Q orthogonal, so that $\kappa(Q) = 1$. This can be achieved by defining $\widetilde{A} := Q^{-1}AQ$ by the real Schur decomposition of A, leading to $a_k = \xi_k$ and $c_k = -\eta_k$ (see e.g. [GL89]). In Chapter 8 this case is elaborated. Here, we analyze a second approach where first $\|\widetilde{Z}^\nu(z)\|$ is minimized and then $\kappa(Q)$. We shall do this for the case where $\widetilde{A}$ is block-diagonal.

7.3.3 The block-diagonal case

In the remainder of this section, we shall analyze the case where $\widetilde{A} := Q^{-1}AQ$ is block-diagonal and we shall use the still free parameters a_k and c_k for reducing the magnitude of $\|\widetilde{Z}^\nu(z)\|$. However, we first justify our choice of a block-diagonal matrix $\widetilde{A}$ by considering the damping of the stiff error components. The following theorem presents a result on the amplification of the stiff iteration errors.

THEOREM 7.4 *Let the conditions of Theorem 7.2 be satisfied and let* $\widetilde{A} := Q^{-1}AQ$ *be block-diagonal. Then, the averaged amplification factor* $\rho^{(\nu)}(z) = \mathcal{O}(z^{(1-\nu)/\nu})$ *as* $z \to \infty$ *and the averaged global amplification factor* $\sigma^{(\nu)}$ *is finite if* $\nu > 2$.

PROOF For $z \to \infty$, it follows from (7.8) that

$$\begin{aligned} Z(z) &= (I - z^{-1}B^{-1})^{-1}(I - B^{-1}A) = Z(\infty) + z^{-1}B^{-1}Z(\infty) + \mathcal{O}(z^{-2}), \\ Z(\infty) &= I - B^{-1}A. \end{aligned}$$

(B may be assumed to be nonsingular because it is required to have positive eigenvalues). More generally, we have that

$$Z^\nu(z) = \sum_{i=1}^{\infty} (Z(\infty))^{\lceil \nu/i \rceil} \mathcal{O}(z^{1-i}),$$

where for any real x, $\lceil x \rceil$ denotes the first integer greater than or equal to x. We first show that all integer powers of $Z(\infty)$ greater than 1 vanish. Since $Z^\nu = Q\widetilde{Z}^\nu Q^{-1}$, we have to show that all integer powers of $\widetilde{Z}(\infty)$ greater than 1 vanish. Because $Q^{-1}AQ$ is block-diagonal, it follows from {(7.10),(7.11)} that T is block-diagonal and from (7.12) that $\widetilde{Z}(z)$ is block-diagonal. Hence, $\widetilde{Z}(\infty)$ is block-diagonal with diagonal blocks $\widetilde{Z}_{kk}(\infty)$. Since by virtue of Theorem 7.2, these blocks have a zero spectral radius, $(\widetilde{Z}_{kk}(\infty))^\nu$ vanishes for $\nu \geq 2$ (this can easily be verified by considering their Schur decompositions). Consequently, $\widetilde{Z}^\nu(\infty)$ itself, and hence $Z^\nu(\infty)$, vanishes for $\nu \geq 2$. From the expansion of $Z^\nu(z)$ we now immediately obtain $Z^\nu(z) = \mathcal{O}(z^{1-\nu})$ as $z \to \infty$. Substitution into {(7.20),(7.21)} yields the result of the theorem. □

From this theorem it follows that the stiff error components may be considered as being removed from the iteration error within two (inner) iterations.

If we only know that $Z(\infty)$ has a zero spectral radius, as in the case of the PDIRK and PTIRK matrices D and L, then $Z^\nu(\infty)$ vanishes for $\nu \geq s$. Hence, by virtue of (7.22) it is seen that for $\nu \geq s$ we have $Z^\nu(z) = \mathcal{O}(z^{1-\lceil \nu/(s-1) \rceil})$ as $z \to \infty$, so that $\rho^{(\nu)}(z) = \mathcal{O}(z^{(1-\lceil \nu/(s-1) \rceil)/\nu})$ as $z \to \infty$ and $\sigma^{(\nu)}$ is finite only if $2(1 - \lceil \nu/(s-1) \rceil)/\nu$ is less than -1, i.e. if $s \leq 2$. Thus, by virtue of the block-diagonality of the matrix $\widetilde{A}$, the PILSRK(T,Q) has a much better stiff initial convergence than the PILSRK(D,I) and PILSRK(L,I) methods.

Reduction of $\|\widetilde{Z}^\nu(z)\|$ in the left-hand half-plane

We derive an estimate for the maximum norm of $\|Z^\nu(z)\|$ in the left-hand half-plane by using the inequality (7.22). Since $\widetilde{A} := Q^{-1}AQ$ is block-diagonal, $\widetilde{Z}^\nu(z)$ is also block-diagonal with diagonal blocks $\widetilde{Z}^\nu_{kk}(z)$ given by

$$\widetilde{Z}_{kk}(z) = \tfrac{z}{(1-u_kz)(1-w_kz)} \cdot \begin{bmatrix} (a_k-u_k)(1-w_kz) & b_k(1-w_kz) \\ (a_k-u_k)v_kz + (c_k-v_k)(1-u_kz) & b_kv_kz + (2\xi_k - a_k - w_k)(1-u_kz) \end{bmatrix}. \tag{7.23}$$

Here, the parameters u_k, v_k and w_k satisfy (7.11). We first minimize the magnitude of $\|\widetilde{Z}^\nu_{kk}(z)\|$. Note that this can be done independently of Q. Having found $\widetilde{Z}_{kk}$, we determine Q by minimizing $\kappa(Q)$. The representation (7.23) suggests setting $a_k = u_k$ and $c_k = v_k$, to obtain for $k \in K$

$$\widetilde{Z}^\nu_{kk}(z) = \zeta^\nu_k(z) \begin{bmatrix} 0 & q_k(z) \\ 0 & 1 \end{bmatrix}, \qquad \zeta_k(z) := \tfrac{(2\gamma\xi_k - \gamma^2\alpha_k - \alpha_k)z}{(1-\gamma\alpha_kz)(\gamma-\alpha_kz)}, \quad q_k(z) := \alpha_k \tfrac{\gamma-\alpha_kz}{c_k}. \tag{7.24}$$

Note that setting $a_k = u_k$ in (7.11) implies $c_k = v_k$.

THEOREM 7.5 *Let the conditions of Theorem 7.4 be satisfied, let $a_k = \gamma\alpha_k$, $|c_k| \geq \gamma^{-1}(1+\gamma^2)\alpha_k$. Then, with respect to the maximum norm, the averaged amplification factor satisfies $\rho^{(\nu)} \leq [\kappa(Q)]^{1/\nu}\rho$, where $\rho = \max\{|1 - 2\gamma(\gamma^2+1)^{-1}\xi_k\alpha_k^{-1}| : k \in K\}$.*

PROOF Let for any matrix $M(z)$ depending on the complex variable z, $|||M|||$ denote the maximum norm of the function $\|M(z)\|$ in the left-hand half-plane, where $\|\cdot\|$ denotes the maximum matrix norm. It is easily seen that

$$|||\widetilde{Z}||| = \max\left\{\left|\frac{2\gamma\xi_k - \gamma^2\alpha_k - \alpha_k}{(1+\gamma^2)\alpha_k}\right|, \left|\frac{2\gamma\xi_k - \gamma^2\alpha_k - \alpha_k}{\gamma c_k}\right| : k \in K\right\}. \tag{7.25}$$

By choosing $|c_k| \geq \gamma^{-1}(1+\gamma^2)\alpha_k$, we find that $|||\widetilde{Z}|||$ equals the asymptotic amplification factor ρ as given in Theorem 7.2. Hence, $|||\widetilde{Z}^\nu||| \leq |||\widetilde{Z}|||^\nu = \rho^\nu$. Obviously, we can never have strict inequality, so that we conclude that $|||\widetilde{Z}^\nu||| = \rho^\nu$. Finally, it follows from (7.22) that $|||Z^\nu||| \leq \kappa(Q)|||\widetilde{Z}^\nu||| = \kappa(Q)\rho^\nu$. Thus, the averaged amplification factor $\rho^{(\nu)}$ is bounded by $[\kappa(Q)]^{1/\nu}\rho$. This completes the proof of the theorem. □

We remark that for $\nu \to \infty$, we have the estimate $\rho^{(\nu)} \leq \rho\max\{[\kappa(S(z))]^{1/\nu} : \mathrm{Re}(z) \leq 0\}$, where $S(z)$ represents the eigensystem of $Z(z)$ and where we assumed that $Z(z)$ has distinct eigenvalues. The advantage of the estimate in Theorem 7.5 is that it holds for all ν.

The transformation matrix Q

In this subsection, we assume that the PILSRK method satisfies the conditions of Theorem 7.5. In order to obtain small amplification factors $\{\rho^{(\nu)}, \sigma^{(\nu)}\}$ as defined by {(7.20,(7.21)}, we shall use the freedom left in choosing the transformation matrix Q. We specify our approach for the case where all eigenvalues $\xi_k \pm \mathrm{i}\eta_k$ of A are complex ($\eta_k \neq 0$), so that $\sigma = s/2$. Then, the column vectors q_j of Q are defined by

$$(q_{2k-1}, q_{2k}) = (\beta_k x_k + \delta_k y_k, -\delta_k x_k + \beta_k y_k)Q_k, \qquad k = 1, \ldots, s/2, \tag{7.26}$$

where β_k and δ_k are free parameters and $x_k \pm \mathrm{i}y_k$ represent the normalized eigenvectors of A corresponding with $\xi_k \pm \mathrm{i}\eta_k$ such that the first component of y_k vanishes. Here, Q_k is a transformation matrix satisfying (cf. (7.9))

$$\begin{aligned} \widetilde{A}_{kk} &= Q_k^{-1} \begin{bmatrix} \xi_k & \eta_k \\ -\eta_k & \xi_k \end{bmatrix} Q_k, \\ \widetilde{A}_{kk} &:= \begin{bmatrix} \gamma\alpha_k & \frac{\gamma(\gamma^2\alpha_k - 2\gamma\xi_k + \alpha_k)}{1+\gamma^2} \\ -\frac{1+\gamma^2}{\gamma\alpha_k} & 2\xi_k - \gamma\alpha_k \end{bmatrix}. \end{aligned} \tag{7.27}$$

It can be verified that the matrix

$$Q_k = \frac{1}{\gamma(\gamma^2\alpha_k - 2\gamma\xi_k + \alpha_k)} \begin{bmatrix} (1+\gamma^2)\eta_k & 0 \\ (1+\gamma^2)(\gamma\alpha_k - \xi_k) & \gamma(\gamma^2\alpha_k - 2\gamma\xi_k + \alpha_k) \end{bmatrix} \tag{7.28}$$

satisfies (7.27). By means of (7.26)–(7.28) it is easily verified that we do obtain the matrix $\widetilde{A} = Q^{-1}AQ$. The advantage of this approach is that the resulting matrix Q has real entries. For a given value of γ, the equations (7.27)–(7.29) determine a family of transformation matrices Q with free parameters vectors $\beta = (\beta_k)$, $\delta = (\delta_k)$ and $c = (c_k)$, where $|c_k| \geq \gamma^{-1}(1+\gamma^2)\alpha_k$.

By a numerical search, we found in the case of the 4-stage and 8-stage Radau IIA correctors for $\gamma = 7/8$ the values (7.29) yielding a sequence of satisfactory small amplification factors $\rho^{(\nu)}$ (see Table 7.2):

$$\begin{aligned} s = 4, \quad & \beta = (5, -4), && \delta = (-1, -5), \\ & a = \tfrac{7}{8}(\alpha_1, \alpha_2), && c = -\tfrac{113}{56}a, \\ s = 8, \quad & \beta = (-0.9, -2, -2, 1.1), && \delta = (1.1, 0.3, 0.3, -1.9), \\ & a = \tfrac{7}{8}(\alpha_1, \alpha_2, \alpha_3, \alpha_4), && c = -\tfrac{113}{56}a. \end{aligned} \tag{7.29}$$

Table 7.2 also lists amplification factors $\rho^{(\nu)}$ for the PILSRK(L,I) and PILSRK(D,I) methods. This table clearly shows that in terms of $\rho^{(\nu)}$-values, the PILSRK(T,Q) methods are superior to the PILSRK(D,I) method. With respect to PILSRK(L,I), the $\rho^{(\nu)}$-values of PILSRK(T,Q) are smaller only for large numbers of inner iterations. In fact, they become less than those associated with PILSRK(L,I) only if ν is greater than about 10. However, in terms of the $\sigma^{(\nu)}$-values, the PILSRK(T,Q) methods are also superior to the PILSRK(L,I) method for small numbers of inner iterations, because in the case of PILSRK(T,Q), $\sigma^{(\nu)}$ becomes finite for $\nu > 2$, whereas PILSRK(L,I) has infinite $\sigma^{(\nu)}$-values for all ν.

TABLE 7.2: *Global amplification factors $\rho^{(\nu)}$ for PILSRK(·,·) methods.*

	4-stage Radau IIA corrector			8-stage Radau IIA corrector		
ν	(D,I)	(L,I)	(T,Q)	(D,I)	(L,I)	(T,Q)
1	3.60	0.59	1.95	19.83	1.03	3.51
2	2.48	0.54	0.98	11.52	0.94	1.88
3	1.64	0.53	0.76	7.74	0.91	1.30
4	1.16	0.53	0.66	5.55	0.90	1.19
5	0.96	0.52	0.61	4.08	0.89	1.09
⋮	⋮	⋮	⋮	⋮	⋮	⋮
9	0.72	0.51	0.53	1.86	0.88	0.87
10	0.69	0.51	0.52	1.72	0.88	0.85
11	0.67	0.51	0.51	1.61	0.88	0.83
⋮	⋮	⋮	⋮	⋮	⋮	⋮
∞	0.52	0.51	0.45	0.90	0.86	0.64

7.4 The Newton–PILSRK iteration process

In actual application of the Newton–PILSRK iteration process {(7.3),(7.4)}, the inner iteration process will not always be iterated to convergence, so that the Newton iterates are only approximately computed. This will affect the convergence and stability behavior and the computational costs of the integration method.

7.4.1 Overall convergence and stability

The overall convergence of the Newton–PILSRK process is determined by the total number of inner iterations summed over all outer iterations in one step, that is, the effective amplification factors associated with the total iteration error $Y^{(j,\nu)} - Y_n$ are approximately given by $\rho^{(i)}$ and $\sigma^{(i)}$, where i denotes the total number of inner iterations needed to compute $Y^{(j,\nu)}$, i.e. $i = (j-1)r + \nu$, and where r denotes the number of inner iterations per outer iteration. In order to see this, we define

$$\begin{aligned} &Y^{(j,0)} := Y^{(j-1,r)}, \qquad G(\Delta) := F(Y+\Delta) - F(Y) - (I \otimes J)\Delta\,, \\ &N := (I - A \otimes hJ)^{-1}(A \otimes I). \end{aligned} \tag{7.30}$$

By a simple manipulation we find that

$$Y^{(j,\nu)} - Y_n = M^{\nu}(Y^{(j-1,r)} - Y_n) + h(I - M^r)NG(Y^{(j-1,r)} - Y_n),\ j=1,\ldots,m, \tag{7.31}$$

where M is defined in (7.7). Ignoring second-order terms, we may set $G(Y^{(j-1,r)} - Y_n) = 0$, to obtain

$$Y^{(j,\nu)} - Y_n = M^i(Y^{(0,r)} - Y_n), \qquad i := (j-1)r + \nu\,. \tag{7.32}$$

TABLE 7.3: *Stable values of mr for* $\gamma = 7/8$.

Iteration method	$s = 4$	$s = 8$
PILSRK(D,I)	7	> 61
PILSRK(L,I)	4	> 43
PILSRK(T,Q)	5	14

From this relation, we see that in a first approximation, the convergence behavior of the Newton–PILSRK iteration process is approximately characterized by the amplification factors. As a consequence, Table 7.2 applies if we replace ν by i.

A second feature of the overall performance of the integration method is its stability if the Newton iterates are not exactly computed. This aspect has been discussed in [HS96], where the number of iterations needed to achieve sufficient stability was computed. The values of mr for which the method becomes and remains L-stable depends on the predictor used. For the extrapolation (EPL) predictor defined by $Y^{(0)} = (P \otimes I)Y_{n-1}$, where P is such that $Y^{(0)}$ has maximal order $q = s - 1$, and the four-stage and eight-stage Radau IIA corrector, these stable mr-values are listed in Table 7.3. In the case of the four-stage corrector, the stable mr-values are acceptable for all three iteration strategies, but for the eight-stage corrector, only the Newton–PILSRK(T,Q) method possesses an acceptable stable mr-value.

Summarizing, we conclude that with respect to the Newton–PILSRK(D,I)-based integration method, the Newton–PILSRK(T,Q) method always generates an integration method that has a superior convergence and stability behavior. With respect to the Newton–PILSRK(L,I)-based integration method, we conclude that the Newton–PILSRK(T,Q) method:

1. damps the stiff error components much stronger for $i < s$ (Theorem 7.4),
2. has a better overall convergence for larger values of i (Table 7.2, with ν replaced by i),
3. is much more stable for the 8-stage corrector (Table 7.3).

7.4.2 Computational costs

In an actual implementation of the linear solver (7.4), we diagonalize (7.4) by a transformation $Y^{(j,\nu)} = (S \otimes I)X^{(j,\nu)}$ to obtain

$$\begin{aligned}(I - S^{-1}BS \otimes hJ)(X^{(j,\nu)} - X^{(j,\nu-1)}) &= -(I - S^{-1}AS \otimes hJ)X^{(j,\nu-1)} \\ &\quad +(S^{-1} \otimes I)C^{(j-1)}, \end{aligned} \tag{7.33}$$

where the matrix $S^{-1}BS$ is diagonal. For the PILSRK(L,I) and PILSRK($T(\gamma \neq 1)$,Q) methods, the matrices $S^{-1}BS$ and S corresponding to the 4-stage and 8-stage Radau IIA

TABLE 7.4: *Total computational costs per step.*

Method	1 processor	$s/2$ processors	s processors
PILSRK(L,I) & PILSRK($T(\gamma \neq 1)$,Q)	$sd(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+4mrsd^2(1 + \frac{s}{2d})$ $+msd(s + C_f - 2d)$	$2d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+8mrd^2(1 + \frac{s}{2d})$ $+2md(2s + C_f - 2d)$	$d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+4mrd^2(1 + \frac{s}{2d})$ $+md(s + C_f - 2d)$
PILSRK($T(1)$,Q)	$sd(\frac{1}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+5mrsd^2(1 + \frac{2s}{5d})$ $+msd(2s + C_f - 2d)$	$2d(\frac{1}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+10mrd^2(1 + \frac{2s}{5d})$ $+2md(2s + C_f - 2d)$	$d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+8mrd^2(1 + \frac{s}{4d})$ $+md(2s + C_f - 2d)$

correctors are given in Appendix B of this chapter. In Appendix A we give a computer program type description of the Newton–PILSRK iteration process {(7.3), (7.4), (7.33)} and a specification of the computational costs of the most important steps of the algorithm. Here, we present in Table 7.4 the total costs per step for s-stage correctors where s is even. In this table, C_f and C_J denote the average costs of one component of f and its Jacobian J, respectively. The following conclusions can be drawn:

1. Newton–PILSRK(L,I) and Newton–PILSRK($T(\gamma \neq 1)$,Q) are equally expensive,

2. If mr is fixed and $d > s + 1/2C_f$, then the costs are minimized for $r = 1$,

3. Newton–PILSRK(L,I) and Newton–PILSRK($T(\gamma \neq 1)$,Q) are to be preferred on s processors, whereas Newton–PILSRK($T(\gamma = 1)$,Q) is to be preferred on one or on σ processors.

7.5 Numerical illustration

In this section, we compare the new Newton–PILSRK($T(7/8)$,Q) method with the Newton–PILSRK(L,I) method. In our experiments, we use the EPL predictor defined in the preceding section and either the 4-stage or the 8-stage Radau IIA corrector with constant stepsizes. We integrated three test problems taken from the CWI test set [LSV96]. In these problems, the initial condition was adapted such that the integration starts outside the transient phase. The first test problem is provided by a problem of Schäfer (called the HIRES problem in [HW91, p. 157]). It consists of 8 mildly stiff nonlinear equations on the interval $[5, 305]$. The second test example is the Pollution problem of Verwer [Ver94]. The ODE system consists of 20 highly stiff nonlinear ODEs on the interval [5,60], originating from an air pollution model. Our third test problem, the Ring Modulator originating from circuit analysis, is a highly stiff system of 15 equations on the interval $[0, 10^{-3}]$, and is due to Horneber [Hor76].

TABLE 7.5: *Newton–PILSRK applied to HIRES with* $h = 15$.

Solver	r	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$	$m=20$
		4-stage Radau IIA corrector					
PILSRK(L,I)	1	*	3.0	4.8	5.1	7.3	7.9
PILSRK(T,Q)		*	4.5	4.9	5.3	7.7	7.9
PILSRK(L,I)	2	*	4.3	4.9	5.3	8.1	7.9
PILSRK(T,Q)		3.9	4.4	4.9	5.4	8.2	7.9
PILSRK(L,I)	10	3.8	4.4	4.9	5.4	8.2	7.9
PILSRK(T,Q)		3.8	4.4	4.9	5.4	8.2	7.9
		8-stage Radau IIA corrector					
PILSRK(L,I)	1	*	*	*	*	8.2	9.9
PILSRK(T,Q)		*	*	*	*	9.3	10.8
PILSRK(L,I)	2	*	*	*	*	9.2	10.1
PILSRK(T,Q)		*	*	5.5	7.0	9.3	10.8
PILSRK(L,I)	10	*	*	5.6	7.0	9.4	10.3
PILSRK(T,Q)		*	*	5.6	7.0	9.3	10.8

The tables of results present the minimal number of correct digits cd of the components of y at the end point of the integration interval (i.e. at the end point, the absolute errors are written as $10^{-\mathrm{cd}}$). Negative cd-values are indicated with $*$. Tables 7.5–7.7 lead us to the following conclusions:

1. For fixed values of $m \geq 3$, the Newton–PILSRK methods always converge and usually find the Newton iterate with high accuracy within two inner iterations (in the case of the 4-stage corrector, we even have convergence for $m \geq 1$).

2. Comparing results for fixed values of mr reveals that $r = 1$ is usually preferable (however, in an actual implementation, m and r should both be determined dynamically, see also the remark in §7.2).

3. Particularly for the 8-stage corrector, the Newton–PILSRK(T,Q) method is more robust than Newton–PILSRK(L,I) for $r \leq 2$, and approximates the Newton iterate usually much better (the better cd-values produced by Newton–PILSRK(L,I) in the Pollution problem for $r = 2$ and $m \in \{3, 4\}$ is due to 'overshoot' and does not mean that Newton–PILSRK(L,I) produces a better approximation to the corrector solution). The divergent behavior is due to the development of instabilities for small values of mr (see Table 7.3).

Finally, we remark that for the relatively difficult Ring Modulator problem, a parallel implementation of the Newton–PILSRK(L,I) method on the four-processor Cray-C98/4256 shows a speed-up ranging from at least 2.4 until at least 3.1 with respect to RADAU5 in one-processor mode (cf. Chapter 4). Since Newton–PILSRK($T(\gamma \neq 1)$,Q) is equally expensive as Newton–PILSRK(L,I), the same speed-ups are expected for Newton–PILSRK($T(\gamma \neq 1)$,Q).

TABLE 7.6: *Newton–PILSRK applied to Pollution problem with* $h = 11$.

Solver	r	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 10$	$m = 20$
		4-stage Radau IIA corrector					
PILSRK(L,I)	1	2.0	3.7	6.3	7.0	10.9	10.9
PILSRK(T,Q)		1.1	5.3	6.9	7.3	10.9	10.9
PILSRK(L,I)	2	4.6	5.7	7.5	8.5	10.9	10.9
PILSRK(T,Q)		4.9	5.7	6.7	7.9	10.9	10.9
PILSRK(L,I)	10	4.6	5.7	6.8	7.9	10.9	10.9
PILSRK(T,Q)		4.6	5.7	6.8	7.9	10.9	10.9
		8-stage Radau IIA corrector					
PILSRK(L,I)	1	*	*	*	*	10.3	10.3
PILSRK(T,Q)		*	*	*	6.7	12.0	12.6
PILSRK(L,I)	2	*	*	*	8.0	10.3	10.7
PILSRK(T,Q)		*	2.9	6.6	7.8	12.6	12.3
PILSRK(L,I)	10	*	4.8	6.7	7.8	11.0	10.9
PILSRK(T,Q)		*	4.8	6.7	7.8	12.5	12.5

TABLE 7.7: *Newton–PILSRK applied to the Ring Modulator with* $h = 1.25 \cdot 10^{-7}$.

Solver	r	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 10$	$m = 20$
		4-stage Radau IIA corrector					
PILSRK(L,I)	1	*	5.7	7.8	8.5	10.2	10.2
PILSRK(T,Q)		*	7.3	8.4	9.7	10.2	10.2
PILSRK(L,I)	2	5.5	7.5	8.7	10.2	10.2	10.2
PILSRK(T,Q)		5.7	7.4	8.8	10.0	10.2	10.2
PILSRK(L,I)	10	5.8	7.4	8.8	9.9	10.2	10.2
PILSRK(T,Q)		5.8	7.4	8.8	9.9	10.2	10.2
		8-stage Radau IIA corrector					
PILSRK(L,I)	1	*	*	*	*	8.6	9.1
PILSRK(T,Q)		*	*	*	10.5	10.8	11.1
PILSRK(L,I)	2	*	*	*	8.9	9.4	9.3
PILSRK(T,Q)		*	8.5	10.4	10.9	10.5	11.3
PILSRK(L,I)	10	*	8.5	9.0	8.9	9.0	9.2
PILSRK(T,Q)		*	8.5	10.4	11.1	11.3	10.6

Appendix A. Costs of PILSRK

In this appendix we specify the costs of the implementations of PILSRK(L,I) and PILSRK($T(\gamma)$,Q). In both methods the iterates satisfy the recursion

$$\begin{aligned}(I - S^{-1}BS \otimes hJ)(X^{(j,\nu)} - X^{(j,\nu-1)}) \;=\; & -(I - S^{-1}AS \otimes hJ)(X^{(j,\nu-1)} - X^{(j-1)}) \\ & -X^{(j-1)} + h(S^{-1}A \otimes I)F(Y^{(j-1)}) \\ & +(E \otimes I)X_{n-1}.\end{aligned}$$

Here, $X_{n-1} = (S^{-1} \otimes I)Y_{n-1}$, $X^{(j,0)} = X^{(j-1)}$, $X^{(0)} = (S^{-1} \otimes I)P(Y_{n-1})$, $X^{(j)} = X^{(j,r)}$, $Y_n = (S \otimes I)X^{(m)}$, $P(\cdot)$ denotes the predictor operator, m the number of outer iterations and r the number of inner iterations. For PILSRK(L,I) and PILSRK($T(\gamma \neq 1)$,Q), the matrix $S^{-1}BS$ is diagonal, for PILSRK($T(1)$,Q), it is block diagonal, with 2×2 lower triangular blocks containing identical diagonal entries. We implemented this recursion as in the following diagram. Here, N is the number of integration steps. The Jacobian is assumed to be updated every time step.

Implementatation of PILSRK.

	$Y_0 = (\mathbf{1} \otimes I)y_0, \quad X_0 = (S^{-1} \otimes I)Y_0$	
	for $n = 1, 2, \ldots, N$	
(s1)	$LU = I - \mathrm{diag}\{S^{-1}BS\} \otimes hJ$	
	$Y^{(0)} = P(Y_{n-1})$	
(s2)	$X^{(0)} = (S^{-1} \otimes I)Y^{(0)}$	
	for $j = 1, 2, \ldots, m$	
(o1)	$R = X^{(j-1)} - h(S^{-1}A \otimes I)F(Y^{(j-1)}) - (E \otimes I)X_{n-1}$	
(o2)	$X_i^{(j,1)} = X_i^{(j-1)} - (LU)_i^{-1}R_i$	(for i odd)
(o3)	$X_i^{(j,1)} = X_i^{(j-1)} - (LU)_i^{-1}(R_i - b_{i,i-1}hJX_{i-1}^{(j,1)})$	(for i even)
	for $\nu = 2, 3, \ldots, r$	
(i1)	$H = (I - S^{-1}AS \otimes hJ)(X^{(j,\nu-1)} - X^{(j-1)}) - R$	
(i2)	$X_i^{(j,\nu)} = X_i^{(j,\nu-1)} - (LU)_i^{-1}H_i$	(for i odd)
(i3)	$X_i^{(j,\nu)} = X_i^{(j,\nu-1)} - (LU)_i^{-1}(H_i - b_{i,i-1}hJX_{i-1}^{(j,\nu)})$	(for i even)
	end	
	$X^{(j)} = X^{(j,r)}$	
(o4)	$Y^{(j)} = (S \otimes I)X^{(j)}$	
	end	
	$Y_n = Y^{(m)}, \quad X_n = X^{(m)}$	
	end	

Notice that for PILSRK(L,I) and PILSRK($T(\gamma \neq 1)$,Q) the matrix $S^{-1}BS$ is diagonal, so that one can omit (**o3**) and (**i3**) for this case, if one performs (**o2**) and (**i2**) for all i. For PILSRK($T(1)$,Q) we only need σ processors to perform the LU-decompositions in parallel, where σ is the number of complex conjugated eigenvalue pairs. Here we assume that s is even, so $\sigma = s/2$.

The following two diagrams list the costs of the most important steps of this algorithm. As before, d is the dimension of the problem. The average costs of one component of the right-hand-side function f and one entry of its Jacobian J are denoted by C_f and C_J, respectively. The Jacobian is assumed to be full. In the first column the computation that has to be performed is listed. The second column gives the number of floating point operations required for this computation if only one processor is available. The sequential costs of the compuation on σ and s processors can be found in the third and fourth column, respectively. For reasons of simplicity, we did not exploit the lower triangular form of the matrix S in PILSRK(L,I), nor the block diagonal form of the matrix $S^{-1}AS$ in PILSRK($T(\gamma)$,Q).

Costs of PILSRK(L,I) & PILSRK($T(\gamma \neq 1)$,Q).

Computation	**Costs (flops)**		
	on 1 processor	on σ processors	on s processors
(s1)	$sd^2(\frac{2}{3}d + \frac{1}{s}C_J + 1)$	$2d^2(\frac{2}{3}d + \frac{1}{s}C_J + 1)$	$d^2(\frac{2}{3}d + \frac{1}{s}C_J + 1)$
(s2)	$2s^2d$	$4sd$	$2sd$
(o1)	$sd(2s + C_f)$	$2d(2s + C_f)$	$d(2s + C_f)$
(o2) ($\forall s$)	$2sd^2$	$4d^2$	$2d^2$
(i1)	$2sd(d + s)$	$4d(d + s)$	$2d(d + s)$
(i2) ($\forall s$)	$2sd^2$	$4d^2$	$2d^2$
(o4)	s^2d	$4sd$	$2sd$
Total per time step	$sd(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+sdm(2d + 3s + C_f$ $+(r-1)(4d + 2s))$	$2d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+2dm(2d + 4s + C_f$ $+(r-1)(4d + 2s))$	$d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+dm(2d + 4s + C_f$ $+(r-1)(4d + 2s))$

Costs of PILSRK($T(1)$,Q).

Computation	**Costs (flops)**		
	on 1 processor	on σ processors	on s processors
(s1)	$sd^2(\frac{1}{3}d + \frac{1}{s}C_J + 1)$	$2d^2(\frac{1}{3}d + \frac{1}{s}C_J + 1)$	$d^2(\frac{2}{3}d + \frac{1}{s}C_J + 1)$
(s2)	$2s^2d$	$4sd$	$2sd$
(o1)	$sd(2s + C_f)$	$2d(2s + C_f)$	$d(2s + C_f)$
(o2)	sd^2	$2d^2$	$2d^2$
(o3)	$2sd^2$	$4d^2$	$4d^2$
(i1)	$2sd(d + s)$	$4d(d + s)$	$2d(d + s)$
(i2)	sd^2	$2d^2$	$2d^2$
(i3)	$2sd^2$	$4d^2$	$4d^2$
(o4)	$2s^2d$	$4sd$	$2sd$
Total per time step	$sd(\frac{1}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+sdm(3d + 4s + C_f$ $+(r-1)(5d + 2s))$	$2d(\frac{1}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+2dm(3d + 4s + C_f$ $+(r-1)(5d + 2s))$	$d(\frac{2}{3}d^2 + \frac{d}{s}C_J + d + 2s)$ $+dm(6d + 4s + C_f$ $+(r-1)(8d + 2s))$

Appendix B. Method parameters

In this appendix we specify the method parameters of the PILSRK(L,I) and PILSRK($T(7/8)$,Q) methods for $s = 4$ and $s = 8$. We list the matrices $S^{-1}BS$ and S, which are needed for the implementation of formula (7.33). As additional information we provide B, the matrix that approximates A.

PILSRK(L,I)

$s = 4$:

$$\text{diag}\{S^{-1}BS\} = \begin{bmatrix} 0.1130 & 0.2905 & 0.3083 & 0.1176 \end{bmatrix}$$

$$S = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ -1.3205 & 1.0000 & 0 & 0 \\ 2.1594 & -27.2263 & 1.0000 & 0 \\ -119.8988 & -66.8265 & 2.3158 & 1.0000 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.1130 & 0 & 0 & 0 \\ 0.2344 & 0.2905 & 0 & 0 \\ 0.2167 & 0.4834 & 0.3083 & 0 \\ 0.2205 & 0.4668 & 0.4414 & 0.1176 \end{bmatrix}$$

$s = 8$:

$$\text{diag}\{S^{-1}BS\} = \begin{bmatrix} 0.0288 & 0.0865 & 0.1345 & 0.1624 & 0.1654 & 0.1427 & 0.0976 & 0.0308 \end{bmatrix}$$

$$S = \begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.0694 & 1.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.0486 & -3.2354 & 1.0000 & 0 & 0 & 0 & 0 & 0 \\ -1.0718 & 7.7636 & -8.1101 & 1.0000 & 0 & 0 & 0 & 0 \\ 1.1852 & -19.0240 & 62.0182 & -88.1175 & 1.0000 & 0 & 0 & 0 \\ -1.4887 & 62.7656 & -0.1720e4 & -0.1141e4 & 11.3694 & 1.0000 & 0 & 0 \\ 2.4708 & -908.4889 & -0.9526e4 & -0.4070e4 & 39.2573 & 4.7028 & 1.0000 & 0 \\ -88.2154 & -2.0073e3 & -1.5590e4 & -0.6097e4 & 58.3751 & 7.4699 & 2.0027 & 1.0000 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0288 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0617 & 0.0865 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0553 & 0.1553 & 0.1345 & 0 & 0 & 0 & 0 & 0 \\ 0.0583 & 0.1424 & 0.2261 & 0.1624 & 0 & 0 & 0 & 0 \\ 0.0567 & 0.1483 & 0.2106 & 0.2619 & 0.1654 & 0 & 0 & 0 \\ 0.0575 & 0.1454 & 0.2171 & 0.2471 & 0.2572 & 0.1427 & 0 & 0 \\ 0.0571 & 0.1467 & 0.2144 & 0.2522 & 0.2460 & 0.2124 & 0.0976 & 0 \\ 0.0573 & 0.1463 & 0.2151 & 0.2510 & 0.2483 & 0.2073 & 0.1338 & 0.0308 \end{bmatrix}$$

PILSRK($T(7/8)$,Q)

$s = 4$:

$$\text{diag}\{S^{-1}BS\} = \begin{bmatrix} 0.1521 & 0.1986 & 0.1737 & 0.2269 \end{bmatrix}$$

$$S = \begin{bmatrix} 2.9526 & 0.3159 & 1.5325 & 0.0276 \\ -7.2663 & -0.8756 & -1.0553 & -0.3113 \\ 3.4202 & 0.9493 & -10.7997 & -2.1349 \\ 34.8970 & 4.3753 & -42.9039 & -5.8960 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.1096 & -0.0430 & 0.0268 & -0.0080 \\ 0.2085 & 0.3064 & -0.0671 & 0.0211 \\ 0.2484 & 0.0823 & 0.2573 & -0.0142 \\ 0.2596 & -0.0515 & 0.4219 & 0.0780 \end{bmatrix}$$

$s = 8$:

$$\text{diag}\{S^{-1}BS\} = \begin{bmatrix} 0.0679 & 0.0886 & 0.0768 & 0.1003 & 0.0823 & 0.1074 & 0.0849 & 0.1109 \end{bmatrix}$$

$$S = \begin{bmatrix} 0.1430 & 0.0149 & 0.0051 & -0.0013 & -0.0208 & -0.0029 & 0.0180 & -0.0001 \\ -0.2667 & -0.0284 & -0.0306 & -0.0006 & 0.0195 & 0.0034 & -0.0182 & -0.0002 \\ 0.4848 & 0.0540 & 0.0915 & 0.0083 & -0.0050 & -0.0010 & 0.0205 & -0.0008 \\ -0.8881 & -0.1065 & -0.0372 & -0.0099 & 0.0975 & 0.0101 & -0.0112 & -0.0072 \\ 1.1326 & 0.1628 & -0.9048 & -0.0996 & 0.2347 & -0.0050 & -0.1102 & -0.0522 \\ 1.6603 & 0.1105 & -0.2681 & 0.0933 & -1.0125 & -0.2481 & -1.3834 & -0.2826 \\ -5.9025 & -0.7539 & 7.6108 & 1.0254 & -7.3467 & -1.0128 & -6.3367 & -0.8981 \\ -8.9828 & -0.9978 & 14.1609 & 1.6360 & -14.1886 & -1.6730 & -12.2810 & -1.4897 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0507 & -0.0264 & -0.0147 & -0.0077 & 0.0061 & -0.0034 & 0.0022 & -0.0008 \\ 0.0295 & 0.0856 & 0.0153 & 0.0162 & -0.0104 & 0.0059 & -0.0037 & 0.0014 \\ 0.0513 & 0.1372 & 0.0952 & -0.0314 & 0.0170 & -0.0096 & 0.0059 & -0.0022 \\ 0.1601 & 0.0455 & 0.0662 & 0.1458 & -0.0342 & 0.0201 & -0.0127 & 0.0048 \\ 0.2072 & 0.0253 & 0.0569 & 0.0462 & 0.1460 & -0.0312 & 0.0131 & -0.0034 \\ 0.2495 & -0.0151 & 0.0590 & 0.0185 & 0.1461 & 0.0202 & 0.0634 & -0.0262 \\ 0.2568 & -0.0281 & 0.0923 & -0.0159 & 0.0405 & 0.0418 & 0.2095 & -0.0688 \\ 0.2653 & -0.0325 & 0.0873 & -0.0924 & 0.1092 & 0.0499 & 0.2190 & -0.0340 \end{bmatrix}$$

Chapter 8

Multistep Runge–Kutta methods

Abstract This chapter 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. With a parallel iterative linear system solver, especially designed for MRK methods, 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 this technique is on s processors equally expensive as the widely used k-step Backward Differentiation Formula on 1 processor, whereas the stability properties are better than that of BDF. A simple implementation of both methods shows that, for the same number of Newton iterations, the accuracy delivered by the new method is higher than that of BDF.

8.1 Introduction

For solving the stiff initial value problem (IVP)

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}}\,, \tag{8.1}$$

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

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

Here, $\otimes$ denotes the Kronecker product, the vector $y^{(n-1)}$ is defined by $(y_{n-k}^{\mathrm{T}}, \ldots, y_{n-1}^{\mathrm{T}})^{\mathrm{T}}$, where y_j approximates the solution at $t = t_j$ and k is the number of previous step points 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 β and the k-dimensional vector κ contain the method parameters. They depend on $h^{(n)}$, which is the vector with k subsequent stepsizes defined by $h^{(n)} := (h_{n-k+1}, \ldots, h_n)^{\mathrm{T}}$. 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 [Pet91] and VODE [BHB92] are based on BDFs. A drawback of BDFs is the loss of stability if the number of step points 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

$$y_n = (\chi^{\mathrm{T}} \otimes I) y^{(n-1)} + h_n (\alpha^{\mathrm{T}} \otimes I) F(Y_n)\,, \tag{8.2}$$

where Y_n is the solution of the equation

$$R(Y_n) = 0, \quad R(Y_n) := Y_n - (G \otimes I)y^{(n-1)} - h_n(A \otimes I)F(Y_n)\,. \tag{8.3}$$

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 $c := (c_1, \ldots, c_s)^{\mathrm{T}}$ 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 vectors α and χ, and the matrices 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 step points, so that they are equally spaced. However, this strategy adds local errors and does not allow good stepsize flexibility, see [Sch94, p. 68].

Stability of MRKs has been investigated for fixed stepsizes in the literature. Even for large values of k, these methods have 'surprisingly' good stability properties [HW91, p. 296]. For example, MRKs of Radau type with $s = 3$ remain stiffly stable for $k \leq 28$ and have modest error constants [Sch94, p. 13].

A drawback of using MRKs is the high cost of solving the non-linear system (8.3) 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

$$(I - A \otimes h_n J_n)(Y_n^{(j)} - Y_n^{(j-1)}) = -R(Y_n^{(j-1)}), \quad j = 1, 2, \ldots, m\,, \tag{8.4}$$

where J_n is the Jacobian of the function f in (8.1) 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 [But76] resulting in

$$\begin{aligned} (I - T^{-1}AT \otimes h_n J_n)(X_n^{(j)} - X_n^{(j-1)}) &= -(T^{-1} \otimes I)R(Y_n^{(j-1)}), \\ Y_n^{(j)} &= (T \otimes I)X_n^{(j)}, \qquad j = 1, 2, \ldots, m. \end{aligned} \tag{8.5}$$

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 [HW96a]. The blocks of the linear system (8.5) are now decoupled, so that the use of σ processors reduces the effective costs to $\frac{16}{3}d^3$, where σ 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 σ processors are available.

By exploiting the special structure of the $2d$-dimensional linear systems in (8.5), it is possible to reduce the costs of solving these systems (see, e.g., [Bin85]). 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

$$\begin{bmatrix} I - \xi_j h_n J_n & -\eta_j h_n J_n \\ \eta_j h_n J_n & I - \xi_j h_n J_n \end{bmatrix}. \tag{8.6}$$

One easily checks that the inverse of (8.6) is

$$(I \otimes \Gamma^{-1}) \begin{bmatrix} I - \xi_j h_n J_n & \eta_j h_n J_n \\ -\eta_j h_n J_n & I - \xi_j h_n J_n \end{bmatrix}, \quad \Gamma = I - 2\xi_j h_n J_n + (\xi_j^2 + \eta_j^2) h_n^2 J_n^2. \tag{8.7}$$

Using σ 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 Γ). On σ 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 chapter, we reduce the implementational costs of MRKs further by following the approach of Chapter 7. Here, the matrix A is approximated by a matrix B with positive distinct eigenvalues and the iterates $Y_n^{(j)}$ in (8.4) are computed by means of the inner iteration process

$$\begin{aligned} (I - B \otimes h_n J_n)(Y_n^{(j,\nu)} - Y_n^{(j,\nu-1)}) &= -(I - A \otimes h_n J_n) Y_n^{(j,\nu-1)} + C_n^{(j-1)}, \\ C_n^{(j-1)} &:= (I - A \otimes h_n J_n) Y_n^{(j-1)} - R(Y_n^{(j-1)}). \end{aligned} \tag{8.8}$$

The index ν runs from 1 to r and $Y_n^{(j,r)}$ is accepted as the solution $Y_n^{(j)}$ of the Newton process (8.4). Furthermore, $Y_n^{(j,0)} = Y_n^{(j-1)}$. Since the matrix B in (8.8) has distinct eigenvalues, applying a similarity transformation Q that diagonalizes B, i.e. $BQ = QD$ where D is a diagonal matrix, leads to:

$$\begin{aligned} (I - D \otimes h_n J_n)(X_n^{(j,\nu)} - X_n^{(j,\nu-1)}) &= -(I - Q^{-1}AQ \otimes h_n J_n) X_n^{(j,\nu-1)} \\ &\quad + (Q^{-1} \otimes I) C_n^{(j-1)}, \quad \nu = 1, \ldots, r. \end{aligned} \tag{8.9}$$

The system (8.9) 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 Chapter 7 we will refer to (8.8) 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 (8.8) converge quickly to the Newton iterates in (8.4). 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 behavior 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 Chapter 7.

The outline of the chapter is as follows. §8.2 briefly describes how to determine the MRK parameters. In §8.3 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 §8.4. Numerical experiments in §8.5 show the performance of the proposed methods on a number of test problems. Finally, we draw some conclusions in §8.6.

8.2 Construction of MRKs

A large class of multistep Runge-Kutta methods consists of multistep collocation methods, which were first investigated by Guillou and Soulé [GS69]. Later, Lie and Nørsett considered the MRKs of Gauss type in [LN89] and Hairer and Wanner [HW91] those of Radau type. In the useful thesis of Schneider [Sch94] 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 [HW91] and [Sch94]. 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 $(u(t_n+c_1h_n)^{\mathrm{T}},\ldots,u(t_n+c_sh_n)^{\mathrm{T}})^{\mathrm{T}}$. In order to compute $u(t)$, we expand it in terms of polynomials ϕ_i and ψ_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, δ_{ij} denotes the Kronecker tensor 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

$$u(t_n+\tau h) = \sum_{j=1}^{k}\phi_j(\tau)y_{n-k+j} + h_n\sum_{j=1}^{s}\psi_j(\tau)u'(t_n+c_jh_n)$$

$$= \sum_{j=1}^{k} \phi_j(\tau) y_{n-k+j} + h_n \sum_{j=1}^{s} \psi_j(\tau) f(u(t_n + c_j h_n)) ,$$

where τ is the dimensionless coordinate $(t - t_n)/h_n$. 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

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

Substituting the first expression into the defining conditions yields

$$\begin{bmatrix} 1 & \tau_1 & \tau_1^2 & \tau_1^3 & \cdots & \tau_1^{s+k-1} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & \tau_k & \tau_k^2 & \tau_k^3 & \cdots & \tau_k^{s+k-1} \\ 0 & 1 & 2c_1 & 3c_1^2 & \cdots & (s+k-1)c_1^{s+k-2} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 1 & 2c_s & 3c_s^2 & \cdots & (s+k-1)c_s^{s+k-2} \end{bmatrix} \begin{pmatrix} d_{0,i}^{\phi} \\ \vdots \\ d_{s+k-1,i}^{\phi} \end{pmatrix} = e_i . \tag{8.10}$$

The matrix of order $s + k$ in (8.10) will be denoted by W. For the polynomials $\psi_i(\tau)$ we derive analogously

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

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

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

Introducing

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

the matrices G and A are respectively given by

$$G = VW^{-1} \begin{bmatrix} e_1 & \cdots & e_k \end{bmatrix} \quad \text{and} \quad A = VW^{-1} \begin{bmatrix} e_{k+1} & \cdots & e_{k+s} \end{bmatrix} .$$

We now construct the abscissae vector c such that we have superconvergence in the step points. Only stiffly accurate Multistep Runge–Kutta methods will be considered,

i.e. $c_s = 1$. This means that we can omit step point formula (8.2) and obtain y_{n+1} from $y_{n+1} = (e_s^{\mathrm{T}} \otimes I)Y_n$. A well known subclass of stiffly accurate MRK methods are the multistep Radau methods, which are A(α)-stable. Their set of collocation points $c_1, \ldots, c_{s-1}$ is given (see [HW91, p. 294]) as the roots in the interval [0,1] of

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

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

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

8.3 Convergence of the inner iteration process

We now discuss the choice of the matrix B in (8.8) 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 (8.4) and (8.8) 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

$$y' = \lambda y, \quad \lambda \in \mathbb{C}\,, \tag{8.11}$$

this recursion reduces to

$$\epsilon_n^{(j,\nu)} = Z(z_n)\epsilon_n^{(j,\nu-1)}, \qquad z_n := h_n\lambda\,. \tag{8.12}$$

Let $\mu(\cdot)$ be the logarithmic norm associated with the Euclidean norm, which can be expressed as $\mu(S) := \frac{1}{2}\lambda_{\max}(S + S^{\mathrm{T}})$, where $\lambda_{\max}(\cdot)$ denotes the algebraically largest eigenvalue of a matrix (see e.g. [HNW93, 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 unfavorable' linear test equation. For the proof of this lemma we refer to [Nev85].

LEMMA 8.1 *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 §8.3.1 and §8.3.2 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_∞ will be referred to as the *stiff amplification matrix.*

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

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

Since all eigenvalues of the matrix B are positive, all the poles of the function $Z(z)$ are in the right-half complex plane. Consequently, $Z^{(j)}(z)$ is analytic in the left-half complex plane and on the imaginary axis. Therefore, we can invoke the maximum principle:

$$\max_{\mathrm{Re}(z_n) \le 0} \|Z^j(z_n)\| = \max_{x_n \ge 0} \|Z^j(\mathrm{i}x_n)\|\,.$$

(It suffices to confine ourselves to the *positive* imaginary axis, because $\|Z(z)\|$ is symmetric with respect to the real axis.) Taking the j^{th} square root on both sides, it follows that

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

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.

8.3.1 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 Chapter 4, 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 $j \ge s$. For reasons that will become clear in §8.3.2, we will refer to this inner iteration process as PILSMRK(L,I).

Table 8.1 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 8.1 shows $\rho^{(j)}(z_n)$ along the imaginary axis $z_n = \mathrm{i}x_n$, $x_n \in \mathbb{R}$, for the PILSMRK(L,I) method with $k = 3$, $s = 4$ and constant stepsizes for $j = 1, 2, 3, 4$ and $j = \infty$. From this figure we clearly see that $\rho_\infty^{(j)} = 0$ for $j \ge s$.

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

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

and plotted $\rho^{(j)}$ as function of ω_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. We do not show these plots here, but they can be summarized by saying 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 $k = 3$, $s = 4$, $\rho^{(j)} \in [0.495, 0.525]$, $\forall j$.

8.3.2 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. The next theorem shows that this leads to a damping of the stiff error components that is optimal in some sense. Since most MRKs of interest have matrices A with at most one real eigenvalue, we restrict ourselves to this class. The theorem makes use of the following definition.

DEFINITION 8.1 *For any $s \times s$ matrix A with at most one real eigenvalue the matrix class $\mathcal{M}_A$ consists of matrices M with the property that there exists an orthogonal matrix U such that*

$$I - M^{-1}A = \begin{cases} U\mathrm{diag}\{\begin{bmatrix} 0 & \vartheta_i \\ 0 & 0 \end{bmatrix}\}U^{\mathrm{T}}, & i = 1, \ldots, s/2, \quad \text{for} \quad s = \text{even}, \\ U\mathrm{diag}\{0, \begin{bmatrix} 0 & \vartheta_i \\ 0 & 0 \end{bmatrix}\}U^{\mathrm{T}}, & i = 1, \ldots, (s-1)/2, \quad \text{for} \quad s = \text{odd}. \end{cases}$$

THEOREM 8.1 *Let A have at most one real eigenvalue. There can be constructed a matrix $B \in \mathcal{M}_A$ for which*

1. $\rho_\infty^{(j)} = 0$ *for* $j > 1$,
2. $\forall\, M \in \mathcal{M}_A : \rho_\infty^{(1)} \leq \|I - M^{-1}A\|_2$.

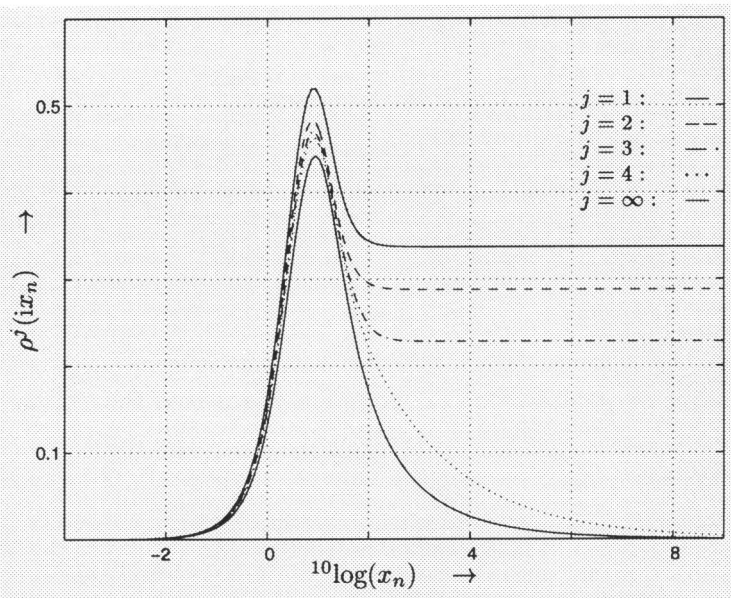

FIGURE 8.1: $\rho^{(j)}(\mathrm{i}x_n)$ *for PILSMRK(L,I) with* $k = 3, s = 4$.

PROOF Since there is freedom in the real Schur form of the matrix A, we first specify how we construct it. Let γ be the vector with eigenvalues of A, and ξ and η be the real and imaginary part of γ, respectively, i.e. $\gamma = \xi + \mathrm{i}\eta$. Order the components of γ as follows (we will motivate this choice later):

$$|\eta_i^2/\xi_i| \geq |\eta_j^2/\xi_j| \quad \text{for} \quad i > j\,. \tag{8.13}$$

In addition, if $|\eta_i^2/\xi_i| = |\eta_{i+1}^2/\xi_{i+1}|$, then $\eta_i > 0$. This sequence is such that, if there is a real eigenvalue, then it has the lowest index in γ and complex eigenvalues are ordered in conjugated pairs by increasing value of η_j^2/ξ_j; the eigenvalue with positive imaginary part comes first within a pair.

Let $e_j^{\mathrm{r}} + \mathrm{i}e_j^{\mathrm{i}}$ be the eigenvector belonging to γ_j, such that $\|e_j^{\mathrm{r}} + \mathrm{i}e_j^{\mathrm{i}}\|_2 = 1$ and $e_{j1}^{\mathrm{i}} = 0$. Define

$$E = \left[\, e_1^{\mathrm{r}} \quad e_1^{\mathrm{i}} \quad e_3^{\mathrm{r}} \quad e_3^{\mathrm{i}} \quad \cdots \quad e_{s-1}^{\mathrm{r}} \quad e_{s-1}^{\mathrm{i}} \,\right],$$

if A has only complex eigenvalues, and

$$E = \left[\, e^{\mathrm{r}} \quad e_2^{\mathrm{r}} \quad e_2^{\mathrm{i}} \quad e_4^{\mathrm{r}} \quad e_4^{\mathrm{i}} \quad \cdots \quad e_{s-1}^{\mathrm{r}} \quad e_{s-1}^{\mathrm{i}} \,\right],$$

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

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

and one block equal to ξ_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:

$$H := (EK)^{-1}A(EK) = K^{-1}(E^{-1}AE)K\,. \tag{8.14}$$

Since K is lower triangular and $E^{-1}AE$ is block diagonal, H is lower block triangular. We now rotate the diagonal blocks of H by means of a matrix Θ given by

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

Here, $j \in \{2, 4, \ldots, s-1\}$ if $\eta_1 = 0$, and $j \in \{1, 3, \ldots, s-1\}$ if $\eta_1 \neq 0$. We will select the angles θ_j such that the second assertion of the theorem will be fulfilled. If we define $S := \Theta^{-1}H\Theta = U^{\mathrm{T}}AU$, with $U := EK\Theta$, then S is the desired Schur form of A. We denote the lower factor of the Crout decomposition of S by L. Setting $B := ULU^{\mathrm{T}}$ yields a stiff amplification matrix that is similar to $I - L^{-1}S$. Consequently, $B \in \mathcal{M}_A$ and Z_∞^j vanishes for $j > 1$, thereby proving the first part of the theorem.

We choose the parameters θ_j such that $\rho_\infty^{(1)} = \max\{|\vartheta_j|\}$ is minimized. A straightforward analysis shows that

$$\vartheta_j = -S_{j,j+1}/S_{j,j}\,, \tag{8.15}$$

where

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

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

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

Using Maple [CGG+91] we established that $|\vartheta_j|$ is minimized for

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

where $\|\cdot\|_F$ denotes the Frobenius norm. Using these values for θ_j in the construction of B leads to the second assertion of the theorem. □

Several remarks should be made with respect to this proof.

- Unlike for the usual eigenvalue problem, where the eigenvalues and eigenvectors are unknown, here we are faced with the problem of computing a real Schur form, given the eigenvalues and eigenvectors of A. The proof serves as a recipe how to construct B. We remark that the construction of the real Schur form is not developed to be cheap, but such that we are able to exploit the freedom in the real Schur form.

- Applying a similarity transformation Q such that $BQ = QD$, we again arrive at scheme (8.9). There is freedom in the choice of the transformation matrix Q that diagonalizes B. If X is a matrix with eigenvectors of B, and Σ and P are diagonal and permutation matrices, respectively, then for every matrix Q of the form

$$Q = X\Sigma P\,, \tag{8.16}$$

  we have that $BQ = QD$. Starting with a fixed matrix X, we determine Σ and P in (8.16) such that the elements of Q and Q^{-1} are not too large.

- Another approach for finding a suitable matrix B, based on rotations that minimize $\rho^{(1)}$, can be found in [HM96].

- 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 (8.13). 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

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

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

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

where

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

TABLE 8.2: *Values of $\rho^{(j)}$ for several PILSMRK(L,U) methods with constant stepsizes.*

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

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 η_k^2/ξ_k come first.

Table 8.2 and Figure 8.2 are the analogues for PILSMRK(L,U) of Table 8.1 and Figure 8.1. In Figure 8.2 we clearly see that $\rho_\infty^{(j)}$ vanishes for $j > 1$. The worst-case $\rho^{(j)}$-values in Table 8.2 are smaller than those in Table 8.1. The difference between PILSMRK(L,I) and PILSMRK(L,U) becomes larger in favor 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 $U^{\mathrm{T}}AU$ with $s(s+1)/2+l$ nonzero entries, where l is the number of complex conjugated eigenvalue pairs, is approximated by $U^{\mathrm{T}}BU$ with $s(s+1)/2$ parameters. The extra price that we have to pay is the construction of the real Schur decomposition of A every time ω_j changes for some j. Since in practice $s \ll d$, we do not consider this as a serious drawback.

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

8.4 Stability

We now investigate the stability of the corrector formula (8.3) and the PILSMRK method given by (8.8) for test equation (8.11) solved with constant stepsizes h. We only consider stiffly accurate methods, i.e. $y_n = (e_s^{\mathrm{T}} \otimes I)Y_n^{(m,r)}$.

Following [Sch94], we write (8.3) in the form

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

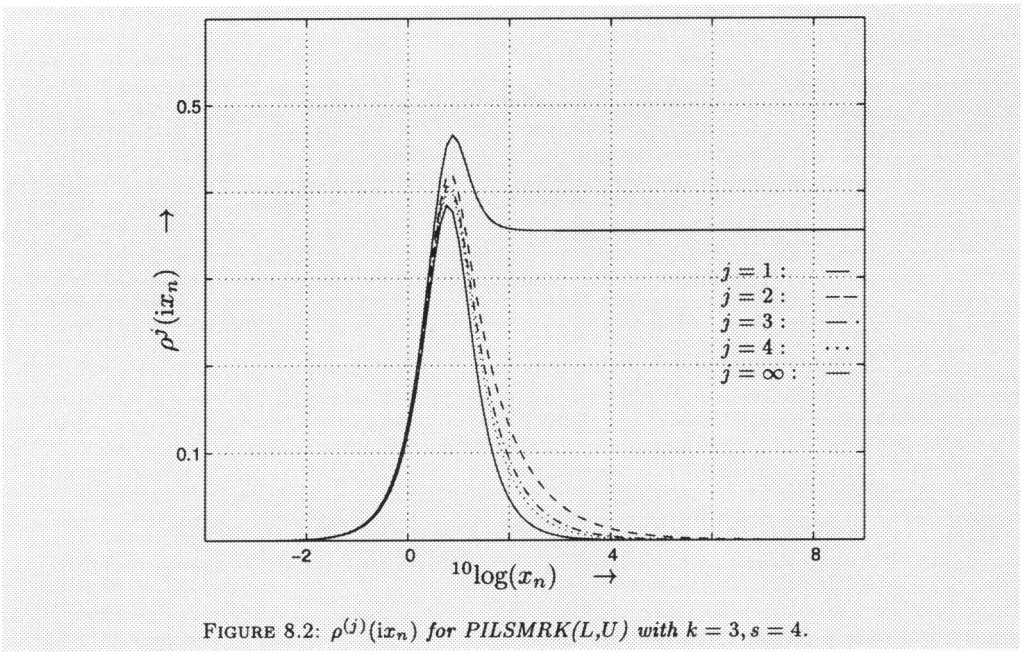

FIGURE 8.2: $\rho^{(j)}(\mathrm{i}x_n)$ *for PILSMRK(L,U) with* $k = 3, s = 4$.

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

$$N = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}.$$

The *stability region* is defined by

$$\mathcal{S} := \{z \in \mathbb{C} \mid \rho(M(z)) < 1\}, \tag{8.17}$$

where $\rho(\cdot)$ denotes the spectral radius function. We use the quantity $\widehat{D}^{(mr)}$ to measure the stability region (see [HW91, 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 step point,

$$Y_n^{(0,r)} = (P \otimes I) Y_{n-1}^{(m,r)}, \tag{8.18}$$

TABLE 8.3: *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

where P is an $s \times s$ matrix. From (8.12) and (8.3) we derive a recursion in ν:

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

An simple 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 (8.18) yields the following recursion in time:

$$Y_n^{(m,r)} = Z^{mr}(z)PY_{n-1}^{(m,r)} + (I - Z^{mr}(z))(I - zA)^{-1}Gy^{(n-1)}\,, \tag{8.19}$$

which we write in the form

$$\begin{pmatrix} y^{(n)} \\ Y_n^{(m,r)} \end{pmatrix} = M^{(mr)}(z) \begin{pmatrix} y^{(n-1)} \\ Y_{n-1}^{(m,r)} \end{pmatrix}, \qquad M^{(mr)}(z) = \begin{bmatrix} M_{11}^{(mr)}(z) & M_{12}^{(mr)}(z) \\ M_{21}^{(mr)}(z) & M_{22}^{(mr)}(z) \end{bmatrix}.$$

From (8.19) 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,

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

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 (8.17) we define the *stability region after mr iterations* by

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

TABLE 8.4: *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

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 8.3 and 8.4 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 (8.18) 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 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 8.3 shows $\mathcal{S}^{(mr)}$ for PILSMRK(L,U) with 3 steps and 4 stages, where $mr \in \{3,5,\infty\}$.

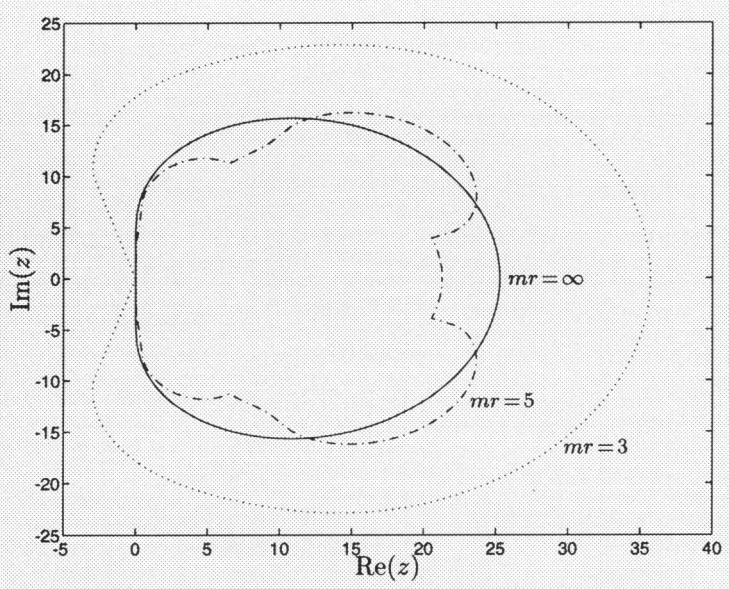

FIGURE 8.3: $S^{(mr)}$ *for PILSMRK(L,U) with* $k = 3, s = 4$.

8.5 Numerical experiments

In this paragraph we study the performance of Newton-PILSMRK numerically for more difficult problems then the linear test problem. We conduct three types of experiments. Firstly, we investigate how many inner iterations PILSMRK needs to find the Newton iterate. For this objective, we implement Newton-PILSMRK with fixed stepsizes, a fixed number of Newton and PILSMRK iterations per step and fixed values of s and k.

Secondly, we compare the Newton-PILSMRK method with a BDF formula using modified Newton. Since we expect that both methods will benefit to the same extent from control strategies (i.e. dynamic Newton iteration strategy, stepsize control, etc.), we again perform this experiment using fixed values of h, s, k, r and m. For a comparison of MRK *codes* with one-step Runge–Kutta codes and BDF codes, we refer to Schneider [Sch94], who gives an excellent overview of this subject.

Finally, the parallel performance of Newton-PILSMRK will be investigated.

Two problems from the 'Test Set for IVP Solvers' [LSV96] are integrated. Our first test example is a problem of Schäfer (called the HIRES problem in [HW91, 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

TABLE 8.5: *Results of PILSMRK(L,I) on test problems.*

			HIRES					Ring Modulator				
s	k	r	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$
2	2	1	3.3	3.7	4.2	4.7	4.9	*	2.8	3.9	3.8	3.8
		2	3.2	3.8	4.3	5.0	4.9	*	3.6	3.8	3.8	3.8
		10	3.2	3.8	4.3	5.0	4.9	*	3.6	3.8	3.8	3.8
	3	1	3.3	3.7	4.2	4.6	5.2	*	3.0	4.1	4.2	4.3
		2	3.2	3.8	4.3	4.8	5.2	*	3.8	4.2	4.3	4.3
		10	3.2	3.8	4.3	4.8	5.2	*	3.8	4.2	4.3	4.3
4	2	1	*	4.6	4.8	5.1	7.3	*	*	6.1	6.5	8.2
		2	*	4.3	4.9	5.3	7.9	*	*	5.8	6.5	8.2
		10	3.7	4.4	4.9	5.4	7.9	*	*	5.8	6.4	8.2
	3	1	*	4.6	4.8	5.1	7.2	*	*	6.1	6.5	8.1
		2	*	4.3	4.9	5.3	7.8	*	*	5.8	6.5	8.1
		10	3.7	4.4	4.9	5.4	7.8	*	*	5.8	6.4	8.1

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 [Hor76] 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 step point value as predictor. We tried extrapolation of more step points, but this did not give satisfactory results for both test problems. The starting values $y_1, y_2, \ldots, y_{k-1}$ 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 $10^{-\mathrm{cd}}$. Negative cd-values are indicated with $*$. The numbers of stages, steps, inner and outer iterations are given by s, k, r and m, respectively.

Tables 8.5 and 8.6 show that the PILSMRK iterates for $r = 1$ are (almost) of the same quality as the Newton iterates, provided that we perform more than 2 Newton iterations.

We also see that the performance of PILSMRK(L,U) is comparable to that of the (L,I) variant. 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 8.1–8.4 indicate that PILSMRK(L,U) can become better than PILSMRK(L,I) for $s > 4$.

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.

From comparing Table 8.7 with Tables 8.5 and 8.6 we learn 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 a fair comparison of BDF with the new method one should take into account the costs of the Butcher transformations as well. Experiments in Chapter 4 (cf. Table 4.9) show that for the ring modulator problem, these costs are less than 10%. Since the sequential costs on s processors are $\mathcal{O}(sd)$ for the transformation costs, $\mathcal{O}(d^2)$ for the forward-back substitutions and $\mathcal{O}(d^3)$ for the LU-decompositions, we expect that the contribution of the transformation costs will rapidly decrease for larger problem dimensions. For tests of the behavior of the linear algebra part on ODEs of higher dimension, we refer to Chapter 6, which studies the linear algebra costs as function of the problem dimension (up to dimension 660) for a method that is comparable to PILSMRK in terms of the solution of linear systems.

TABLE 8.6: *Results of PILSMRK(L,U) on test problems.*

			HIRES					Ring Modulator				
s	k	r	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$
2	2	1	3.3	3.8	4.2	4.8	4.9	*	2.8	3.9	3.8	3.8
		2	3.2	3.8	4.3	5.0	4.9	*	3.6	3.8	3.8	3.8
		10	3.2	3.8	4.3	5.0	4.9	*	3.6	3.8	3.8	3.8
	3	1	3.3	3.8	4.2	4.7	5.2	*	3.1	4.1	4.3	4.3
		2	3.2	3.8	4.3	4.8	5.2	*	3.8	4.2	4.3	4.3
		10	3.2	3.8	4.3	4.8	5.2	*	3.8	4.2	4.3	4.3
4	2	1	*	*	4.9	5.1	7.2	*	*	5.8	6.3	8.2
		2	2.6	4.4	4.9	5.4	7.9	*	*	5.8	6.4	8.2
		10	3.7	4.4	4.9	5.4	7.9	*	*	5.8	6.4	8.2
	3	1	*	*	4.9	5.2	7.2	*	*	5.8	6.3	8.1
		2	3.6	4.4	4.9	5.4	7.8	*	*	5.8	6.4	8.1
		10	3.7	4.4	4.9	5.4	7.8	*	*	5.8	6.4	8.1

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 C98/4256 and integrated the ring modulator, using again 4000 constant integration steps. The Cray C98/4256 is a shared memory computer with four processors. Table 8.8 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 [Cra94b] 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 8.8.

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 [Cra94a] for the specification of these settings.

TABLE 8.7: *Results of BDF on test problems.*

			HIRES					Ring Modulator				
s	k	r	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$	$m=1$	$m=2$	$m=3$	$m=4$	$m=10$
1	2	1	2.9	3.5	3.1	3.0	3.0	1.1	1.1	1.1	1.1	1.1
	3	1	2.8	3.7	3.6	3.4	3.3	1.6	1.5	1.6	1.6	1.6
	4	1	2.8	3.4	4.4	3.8	3.6	1.8	1.9	1.9	1.9	1.9
	5	1	2.7	3.3	4.2	4.1	3.8	2.4	2.9	2.9	2.9	2.9
	6	1	2.8	3.4	4.1	3.9	3.7	2.4	*	2.9	*	2.9

From Table 8.8 we conclude that the Newton-PILSMRK methods have a satisfactory parallel performance. With respect to the scalability of the method, we remark that the number of processors involved equals the number of stages s. Since the step point order is given by $2s + k - 2$, using more than four processors leads to an order that might be too high for most practical applications. Therefore, we aim at two or four processors, which are natural numbers for many computer architectures.

TABLE 8.8: *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

8.6 Summary and conclusions

In this chapter, 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 Newton-PILSMRK method with k-steps and s-stages is equally expensive on s processors as the k-step BDF on 1 processor, whereas the order is higher and the stability properties are better than that of BDF.
- Tests with implementations of Newton-PILSMRK and BDF without control strategies on two problems from the CWI test set show that for the same number of sequential function evaluations, Newton-PILSMRK delivers higher accuracies than BDF, although Newton-PILSMRK did not solve the corrector equation entirely.
- Increasing the number of previous step points k, leads to a better convergence behavior 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 behavior, but its stability properties are worse for $s \leq 4$.

Appendix

In this appendix we list the parameters c, G and A in (8.3) 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 δ and Q, where δ =diag(D) and D, Q are the matrices in (8.9), for both the Crout approach PILSMRK(L,I) and the Schur–Crout approach PILSMRK(L,U).

$s = 2,\ k = 2$:

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

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

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

Crout:

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

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

Schur–Crout:

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

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

$s = 2,\ k = 3$:

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

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

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

Crout:

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

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

Schur–Crout:

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

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

$s = 4,\ k = 2$:

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

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

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

Crout:

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

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

Schur–Crout:

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

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

$s = 4,\ k = 3$:

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

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

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

Crout:

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

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

Schur–Crout:

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

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

Chapter 9
Estimating the error

Abstract For implicit Runge–Kutta methods intended for stiff ODEs or DAEs, it is often difficult to embed a local error estimating method which gives realistic error estimates for stiff/algebraic components. If the embedded method's stability function is unbounded at $z = \infty$, stiff error components are grossly overestimated. In practice some codes 'improve' such inadequate error estimates by premultiplying the estimate by a 'filter' matrix which damps or removes the large, stiff error components. Although improving computational performance, this technique is somewhat arbitrary and lacks a sound theoretical backing. In this chapter, we resolve this problem by introducing an *implicit* error estimator. It has the desired properties for stiff/algebraic components without invoking artificial improvements. The error estimator contains a free parameter which determines the magnitude of the error, and we show how this parameter is to be selected on the basis of method properties. The construction principles for the error estimator can be adapted to all implicit Runge–Kutta methods, and a better agreement between actual and estimated errors is achieved, resulting in better performance.

9.1 Introduction

We shall consider the problem of estimating the local error in a single step when an implicit Runge–Kutta method (IRK) is applied to a stiff system of ordinary differential equations

$$y'(t) = f(y(t)), \qquad y(t_0) = y_0, \qquad y, f \in \mathbb{R}^d, \qquad t_0 \le t \le t_{\text{end}} . \tag{9.1}$$

Using standard notation, [HW96b], we write an s–stage IRK (A, b) in the form

$$Y = \mathbb{1} \otimes y_n + h(A \otimes I)F(Y) , \tag{9.2}$$

$$y_{n+1} = y_n + h(b^{\mathrm{T}} \otimes I)F(Y) , \tag{9.3}$$

where y_n approximates $y(t_n)$. Furthermore, h is the stepsize, Y is the sd–dimensional stage vector whose s component stage vectors Y_i approximate $y(t_n + c_i h)$. The abscissae are defined by $c = A\mathbb{1}$, with $\mathbb{1} = (1, 1, \ldots, 1)^{\mathrm{T}}$. Finally, F stands for the component–wise evaluation of f, i.e.

$$F(Y) = (f(Y_1)^{\mathrm{T}}, f(Y_2)^{\mathrm{T}}, \ldots, f(Y_s)^{\mathrm{T}})^{\mathrm{T}}.$$

By solving the nonlinear system (9.2) we obtain Y and compute y_{n+1} from (9.3). (Here we leave aside the option of solving for the stage derivatives $F(Y)$.)

The primary means to control the accuracy of the computational process is to vary the stepsize. In order to do this we need estimates of the error committed in each individual

step, the *local error*. Let $\hat{y}(t;\tau,\eta)$ denote a solution to the differential equation with initial value $y(\tau) = \eta$. Then the local error in y_{n+1} is $\epsilon = y_{n+1} - \hat{y}(t_{n+1};t_n,y_n)$. The quantity ϵ is estimated by computing a second approximation, $\hat{y}_{n+1}$, to $\hat{y}(t_{n+1};t_n,y_n)$. In embedded IRK methods, this is obtained by taking another linear combination $\hat{b}$ of the stage derivatives. In the sequel we will use the following definition. If $\epsilon = \mathcal{O}(h^p)$, then y_{n+1} is said to be of *local order p*.

9.2 Error estimation in RADAU5

Because of difficulties in finding $\hat{b}$ such that the order of the error estimate is suitable, one may have to introduce extra parameters. Let us consider the widely used Radau IIA methods, [HW96b, p. 123], where the following formula for $\hat{y}_{n+1}$ is used:

$$\hat{y}_{n+1} = y_n + h\left(\hat{b}_0 f(y_n) + (\hat{b}^{\mathrm{T}} \otimes I)F(Y)\right). \tag{9.4}$$

Here $\hat{b}_0$ is a free parameter and $\hat{b}$ is an s–dimensional vector, which is determined such that $\hat{y}_{n+1}$ is of local order $s+1$, i.e., $\hat{b}$ must satisfy the order conditions

$$C\,\hat{b} = (1-\hat{b}_0, 1/2, 1/3, \ldots, 1/s)^{\mathrm{T}}.$$

The $s \times s$ matrix C has entries $c_{ij} = c_j^{i-1}$. Note that putting $\hat{b}_0 = 0$ in (9.4) would by the order conditions lead to the same formula as (9.3). Consequently, $\hat{b}_0 \neq 0$; at least one extra parameter is necessary to obtain a nonzero error estimate.

The estimate ϵ is now computed as

$$\epsilon = y_{n+1} - \hat{y}_{n+1}\,, \tag{9.5}$$

and y_{n+1} is accepted as an approximation to $y(t_{n+1})$ if $\|\epsilon\|$ is less than the specified tolerance. As ϵ is dependent on the stepsize, its ratio to the tolerance is also used to compute the next stepsize.

Most IRKs are constructed in such a way that they are at least A–stable. However, the reference formula (9.4) is normally not A–stable. Consequently, $\|\epsilon\|$ can be very large due to large stiff error components. In practice this is typically the case, since IRK methods are indeed intended to solve stiff problems or DAEs.

In RADAU5, [HW96a], which is an implementation of the 3–stage Radau IIA method, Hairer and Wanner use the following remedy, [HW96b, p. 123], which is attributed to Shampine [SB84]. A modified error estimate $\hat{\epsilon}$ is constructed from

$$\hat{\epsilon} = (I - \gamma h J)^{-1}\epsilon\,, \tag{9.6}$$

in which $\hat{y}_{n+1}$ is computed from (9.4) with $\hat{b}_0 = \gamma$, the single real eigenvalue of A. The matrix $(I - \gamma h J)^{-1}$ is then available and factorized from the Newton iteration used to solve (9.2). To see the effect of this transformation, consider the test equation $y' = \lambda y$; we now have $\hat{\epsilon} \to 1$ as $h\lambda \to \infty$, as opposed to $\epsilon \to \infty$. The purpose of the premultiplication by $(I - \gamma h J)^{-1}$ is thus to keep the error estimate bounded also for large values of h by filtering out stiff error components.

9.3 Case study: The implicit Euler method

The filtering technique has also been used in other contexts where it has a theoretical foundation in terms of the map from a residual to the corresponding error. In the context above, however, it is a trick—albeit a necessary one—in order to restore the full potential of the Radau IIA method.

In order to see where and how the filtering is justified, we consider the simplest Radau IIA method, i.e. the implicit Euler method

$$y_{n+1} = y_n + hf(y_{n+1})\,. \tag{9.7}$$

If we insert the local solution $\hat{y}(t; t_n, y_n)$ into this discretization, there results a defect, or *local residual* δ:

$$\hat{y}(t_{n+1}) = y_n + hf(\hat{y}(t_{n+1})) - \delta\,. \tag{9.8}$$

We find the *local error* $\epsilon = y_{n+1} - \hat{y}(t_{n+1}; t_n, y_n)$ by subtracting (9.8) from (9.7) and obtain an algebraic relation between the residual and the error:

$$\epsilon = hf(\hat{y}(t_{n+1}) + \epsilon) - hf(\hat{y}(t_{n+1})) + \delta\,. \tag{9.9}$$

Linearizing and solving for ϵ we obtain the error/residual relation,

$$\epsilon = (I - hJ)^{-1}\delta\,. \tag{9.10}$$

This equation is the mathematical justification of 'filtering'. As is well–known, there is an important conceptual as well as numerical difference between a residual and its corresponding error—the defect and error are elements of different spaces. Although this equation is well established, [HNW93, p. 369], it is frequently overlooked. The reason seems to be an overemphasis on asymptotics; as $hJ \to 0$ we have $\epsilon \approx \delta$, i.e. in the non-stiff case it does not matter if one estimates ϵ or δ, but in the stiff case the difference is known to be very significant. This observation has led to the view that a 'poor' error estimate can be improved by the premultiplication of a filtering matrix. Even if this works in practice, such arbitrariness in error estimation ought to be replaced by a search for qualitatively correct error estimates. Note that in embedded IRK methods, filtering is in principle *never* justified since one normally estimates a local error, never a local residual. The situation may, however, be different for defect estimation.

In the next section we suggest an error estimate which has an inherent damping of stiff error components as a design criterion. No extra filtering is required or permitted (as it cannot be justified). As a starting point we note that the poor asymptotic behavior of ϵ as defined by (9.4) is caused by (9.4) being essentially an explicit formula. Thus, $\hat{y}_{n+1}$ is computed from old data, the stage derivatives *and the explicitly calculated* $hf(y_n)$. This turns the error estimator formula effectively into an explicit method, and consequently all hopes for a proper behavior for large values of h are in vain.

9.4 An implicit error estimate

Instead of (9.4) we propose to use an implicit reference formula of the structure

$$\hat{y}_{n+1} = y_n + h\left(\hat{b}_0 f(y_n) + (\hat{b}^{\mathrm{T}} \otimes I)F(Y) + \gamma f(\hat{y}_{n+1})\right), \tag{9.11}$$

where γ is such that $(I - \gamma hJ)^{-1}$ is available from the (transformed) Newton process used to solve for Y from (9.2). Solving $\hat{y}_{n+1}$ from (9.11) by a modified Newton process leads to the recursion

$$\begin{aligned} r^{(j)} &= \hat{y}_{n+1}^{(j)} - y_n - h\left(\hat{b}_0 f(y_n) + (\hat{b}^{\mathrm{T}} \otimes I)F(Y) + \gamma f(\hat{y}_{n+1}^{(j)})\right), \\ \hat{y}_{n+1}^{(j+1)} &= \hat{y}_{n+1}^{(j)} - (I - \gamma hJ)^{-1} r^{(j)}. \end{aligned}$$

The natural starting value is $\hat{y}_{n+1}^{(0)} = y_{n+1}$. Since we are computing an error estimate we do not need high accuracy and may consider the first Newton iterate $\hat{y}_{n+1}^{(1)}$ as *the* reference formula itself. This yields

$$\hat{y}_{n+1}^{(1)} = y_{n+1} + h(I - \gamma hJ)^{-1}\left(\left((\hat{b}^{\mathrm{T}} - b^{\mathrm{T}}) \otimes I\right) F(Y) + \hat{b}_0 f(y_n) + \gamma f(y_{n+1}\right).$$

In this formula, we determine $\hat{b}$ such that $\hat{y}_{n+1}^{(1)}$ is of local order $s+1$, which means that we require

$$C\,\hat{b} = (1 - \hat{b}_0, 1/2, 1/3, \ldots, 1/s)^{\mathrm{T}} - \gamma \mathbb{1}. \tag{9.12}$$

The parameter $\hat{b}_0$ is free but required to be nonzero in the case $c_s = 1$, as taking $\hat{b}_0 = 0$ then yields $\hat{y}_{n+1} \equiv y_{n+1}$.

For methods with $c_s = 1$ we have $C^{-1}\mathbb{1} = e_s$ and

$$\hat{y}_{n+1}^{(1)} = y_{n+1} + h(I - \gamma hJ)^{-1}\left((-\hat{b}_0 e_1^{\mathrm{T}} C^{-\mathrm{T}} \otimes I)F(Y) + \hat{b}_0 f(y_n)\right). \tag{9.13}$$

Consequently, the *error estimator formula*

$$\begin{aligned} \epsilon &= y_{n+1} - \hat{y}_{n+1}^{(1)} \\ &= \hat{b}_0 h(I - \gamma hJ)^{-1}\left((e_1^{\mathrm{T}} C^{-\mathrm{T}} \otimes I)F(Y) - f(y_n)\right) \end{aligned} \tag{9.14}$$

becomes a homogeneous function of $\hat{b}_0$. In other words, the choice of $\hat{b}_0$ determines the magnitude of the error estimate.

In general, we define the error estimator formula by

$$\begin{aligned} \epsilon &= y_{n+1} - \hat{y}_{n+1}^{(1)} \\ &= h(I - \gamma hJ)^{-1}\left(\left((b^{\mathrm{T}} - \hat{b}^{\mathrm{T}}) \otimes I\right) F(Y) - \hat{b}_0 f(y_n) - \gamma f(y_{n+1})\right). \end{aligned} \tag{9.15}$$

Now consider the test equation $y' = \lambda y$, for which

$$\hat{y}_{n+1}^{(1)} = \hat{R}^{(1)}(z) y_n \,, \qquad z := h\lambda \,.$$

The value of $\hat{R}^{(1)}(\infty)$ of the reference formula is known to be of relevance to the size of the estimated error in the stiff components. Although this is not a matter of stability, it is desirable that $\hat{R}^{(1)}(\infty)$ is fairly small. A straightforward derivation yields

$$\hat{R}^{(1)}(z) = R(z) + \frac{z}{1-\gamma z}\left((\hat{b}^{\mathrm{T}} - b^{\mathrm{T}})(I - zA)^{-1}\mathbb{1} + \hat{b}_0 + \gamma R(z)\right), \tag{9.16}$$

where $R(z)$ is the stability function of the implicit Runge–Kutta method. If A is non-singular, we thus obtain

$$\lim_{z\to\infty} \hat{R}^{(1)}(z) = -\frac{\hat{b}_0}{\gamma} \,. \tag{9.17}$$

Thus the stiff error components are damped if $|\hat{b}_0/\gamma| < 1$. This damping is desirable as the error estimator will 'see' a stiff error component from the previous step's iteration error multiplied by $|\hat{b}_0/\gamma|$.

For an s–stage Radau IIA method one can easily give an explicit formula for our new error estimator. If we write $R(z) = P_R(z)/Q_R(z)$ and normalize P_R and Q_R such that $Q_R(z)$ is monic (e.g. for $s = 3$ we have $Q_R(z) = z^3 - 9z^2 + 36z - 60$), then by (9.16), $R(z) - \hat{R}^{(1)}(z)$ is a rational function with denominator $(1-\gamma z)Q_R(z)$. Thus the degree of the denominator is $s+1$. By (9.17) the numerator then has degree at most $s+1$. As the local order of the error estimator is $s+1$, however, the numerator only contains a single power of z, viz. z^{s+1}. It follows that

$$R(z) - \hat{R}^{(1)}(z) = -\frac{\hat{b}_0 z^{s+1}}{(1-\gamma z)Q_R(z)} \,. \tag{9.18}$$

For the 3–stage Radau IIA, $R(z) - \hat{R}^{(1)}(z)$ thus has a four–fold zero at $z = 0$ and the same poles as $R(z)$ with the exception that $z = 1/\gamma$ is a double pole.

REMARKS Note that if $b^{\mathrm{T}} = e_s^{\mathrm{T}} A$, where e_s is the s^{th} canonical basis vector of $\mathbb{R}^s$, then (9.15) and (9.6) differ only by a factor $\hat{b}_0/\gamma$. The condition $b^{\mathrm{T}} = e_s^{\mathrm{T}} A$ ('stiff accuracy'), holds for all Radau IIA methods as well as for the Lobatto IIIA and IIIC methods. For these methods our implicit error estimator justifies filtering by providing an estimate with the same effect. For other methods, however, one must be more careful. Thus e.g., it is incorrect to use filtering for the implicit midpoint method, which is a Gauss method, but harmless to use it for the trapezoidal rule, which falls into the Lobatto IIIA category. In order to avoid mistakes, we suggest that the construction of implicit estimators is considered to be the normal route instead of filtering. Finally we remark that even in cases when the error estimator formula is not a homogeneous function of $\hat{b}_0$, we may select the magnitude of the error estimator with a multiplicative factor; we may consider

$E(z) = \theta(R(z) - R^{(1)}(z))$ as the error estimator, where the parameter θ is to be carefully determined so that the estimator gives a proper approximation to the actual error. This technique may be of particular importance for DAEs. ◇

9.5 Choosing $\hat{b}_0$

We shall finally discuss the choice of the free parameter $\hat{b}_0$, and limit ourselves to methods with $c_s = 1$ such as Radau IIA methods. Specifically, we will motivate a suitable choice of $\hat{b}_0$ for the 3–stage, 5^{th} order Radau IIA method. For such methods the new error estimator is a homogeneous function of $\hat{b}_0$, i.e. $\hat{b}_0$ determines the magnitude of the error estimate.

Today it is common to use 'local extrapolation', i.e. the embedded reference formula has a lower order than the method itself. Therefore it is not possible to choose $\hat{b}_0$ such that the estimator fits the actual error, but it is still important that the estimator has the right order of magnitude.

We argue that the most important design goal is that the error estimator does not significantly underestimate the error. On the other hand, a too large value of $\hat{b}_0$ will degrade performance. A small value is also desirable to reduce $\hat{R}^{(1)}(\infty)$. To find a suitable value, we model the error of the Radau IIA method by first considering the linear test equation with $z = h\lambda$. The *relative local error*, defined by

$$\left| \frac{y_{n+1} - \hat{y}(t_{n+1}; t_n, y_n)}{y_n} \right| = |R(z) - \mathrm{e}^z|,$$

is investigated on two domains: $\mathcal{A}$, where the method operates in its asymptotic regime, and $\mathcal{B}$, where the method is able to yield accurate results. $\mathcal{B}$ is considerably larger than $\mathcal{A}$.

Obviously $\mathcal{A}$ and $\mathcal{B}$ must contain a neighborhood of the origin. We take $\mathcal{A}$ to be a disk of radius ρ,

$$\mathcal{A}(\rho) = \{z \in \mathbb{C} : |z| \leq \rho\}.$$

The selection of the radius is based on several criteria. First, $\mathcal{A}(\rho)$ must exclude the poles of $R(z)$ which for the 3–stage Radau IIA are located at $1/\gamma \approx 3.6378$ and $2.6811 \pm 3.0504\mathrm{i}$, respectively. By (9.18), the poles of the reference formula $\hat{R}^{(1)}(z)$ are then also excluded. Furthermore, $\mathcal{A}(\rho)$ should cover the central portion of the order star of the method, [IN91, p. 7], as this corresponds to the region of the complex plane where the Padé approximation $R(z)$ is close to e^z. Last, the intersection with the imaginary axis is an important criterion of relevance for oscillatory systems. To resolve an angular frequency of ω, the stepsize must satisfy $h\omega < \pi$ by the sampling theorem. In practice, however, the numerical method is unable to accurately resolve this frequency with stepsizes exceeding $h\omega = \pi/2$. Based on these considerations, we have taken $\rho = \pi/2$. The selected asymptotic domain $\mathcal{A}(\pi/2)$ meets all the criteria above.

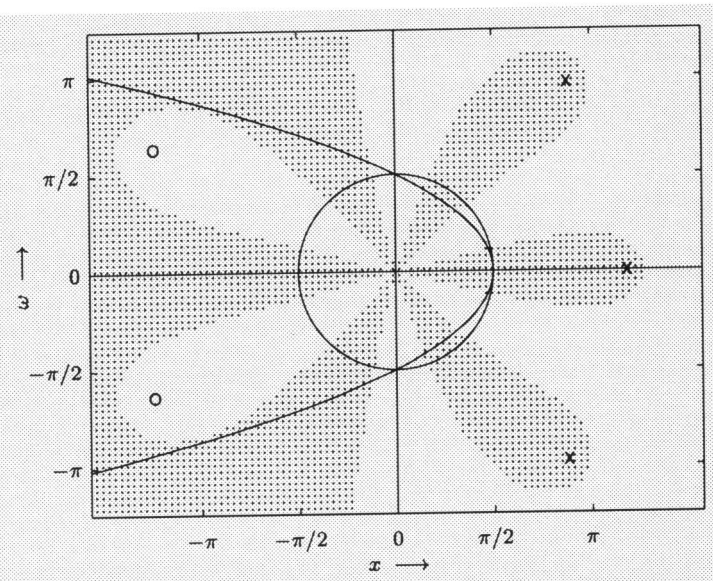

FIGURE 9.1: *Plots of $\partial\mathcal{A}(\pi/2)$ and $\partial\mathcal{B}(\pi/2)$, together with the order star of the 3–stage Radau IIA method. Poles and zeros of $R(z)$ are denoted by $\times$ and $\circ$, respectively.*

$\mathcal{B}(\rho)$ should contain most of $\mathcal{A}(\rho)$ as well as a large portion of the negative half-plane. Again, high frequencies cannot be resolved, but $\mathcal{B}(\rho)$ should cover the negative real axis if the method—like the Radau IIA—is able to produce accurate solutions there. We have chosen to consider the parabolic domain

$$\mathcal{B}(\rho) = \{z = x + \mathrm{i}\omega \,:\, x \leq (\rho - \omega)(\rho + \omega)/\rho\},$$

and $\mathcal{A}(\pi/2)$, $\mathcal{B}(\pi/2)$ and the order star of $R(z)$ are plotted in Figure 9.1. The Radau IIA method is able to provide reasonable accuracy inside $\mathcal{B}(\pi/2)$. The method is still of use in large portions of the complex plane outside $\mathcal{B}(\pi/2)$, e.g. in all of $\mathbb{C}^-$; A–stability implies that $|R(z)| \leq 1$ on $\mathbb{C}^-$ just like $|\mathrm{e}^z| \leq 1$, even if the relative local error $|R(z) - \mathrm{e}^z|$ cannot be considered to be 'small' on all of $\mathbb{C}^-$.

As $R(z) - \mathrm{e}^z$ is an analytic function in the domain of accuracy $\mathcal{B}(\pi/2)$, $\max|R(z) - \mathrm{e}^z|$ is attained on $\partial\mathcal{B}(\pi/2)$ by virtue of the maximum modulus theorem. Thus we find that $\max|R(z) - \mathrm{e}^z| = 0.067$ in $\mathcal{B}(\pi/2)$, and we may choose $\hat{b}_0$ (i.e. the magnitude of the error estimator) so that $\max|R(z) - \hat{R}^{(1)}(z)|$ comes close to the maximum of the actual error. This suggests choosing $\hat{b}_0/\gamma = 0.067$, or $\hat{b}_0 \approx 0.018$, and in Figure 9.2 (left), we plot $|R(z) - \mathrm{e}^z|$ and the error estimator $|R(z) - \hat{R}^{(1)}(z)|$ on $\partial\mathcal{B}(\pi/2)$ for $\hat{b}_0 = 0.02$. Because the maxima may not occur at the same points, we verify in Figure 9.2 (right) that the error estimator with its chosen magnitude does not exhibit any significant underestimation of the error on the negative real axis.

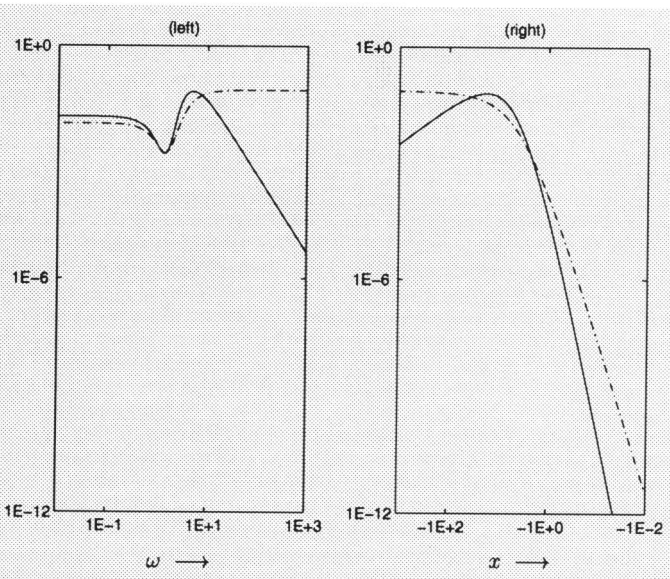

FIGURE 9.2: *Plots of relative error* $|R(z)-\mathrm{e}^z|$ *(solid) and error estimator* $|R(z)-\hat{R}^{(1)}(z)|$ *(dash-dotted) on* $\partial\mathcal{B}(\pi/2)$ *for* $10^{-2} \le \omega \le 10^3$ *(left) and on the negative real axis for* $-10^3 \le x \le -10^{-2}$ *(right). The plot on the right clearly shows slopes of* -6 *and* -4 *when* $|x| < 1$*, and a slight underestimation of the error near* $x = -10$*. The plots were obtained using* $\hat{b}_0 = 0.02$ *in order to match the levels of the error and its estimate.*

To investigate the new estimator in the asymptotic regime, we have plotted $|R(z)-\mathrm{e}^z|$ and $|R(z) - \hat{R}^{(1)}(z)|$ on $\partial\mathcal{A}(\pi/2)$ in Figure 9.3 (left), showing that their magnitudes are similar there. Note that because the error estimator has lower order than the method, it is still likely to significantly overestimate the error at sharp tolerances. This is seen in Figure 9.3 (right), where we study the ratio

$$K(z) = \frac{R(z) - \mathrm{e}^z}{R(z) - \hat{R}^{(1)}(z)} \tag{9.19}$$

and have plotted

$$k(\rho) = \max_{|z|\le\rho} |K(z)| \tag{9.20}$$

for $0 < \rho \le \pi/2$. The plot suggests that the error estimator underestimates the error outside $\mathcal{A}(1.3)$. This underestimation is benign, however, as verified by Figure 9.4, which shows the level curves $|K(z)| = \kappa$ for $\kappa = 0.2(0.2)1.2$. Thus, the underestimation occurs only in the right half-plane for $|z| > 1.3$, where, in the absence of dissipativity, the method is less likely to proceed with large steps.

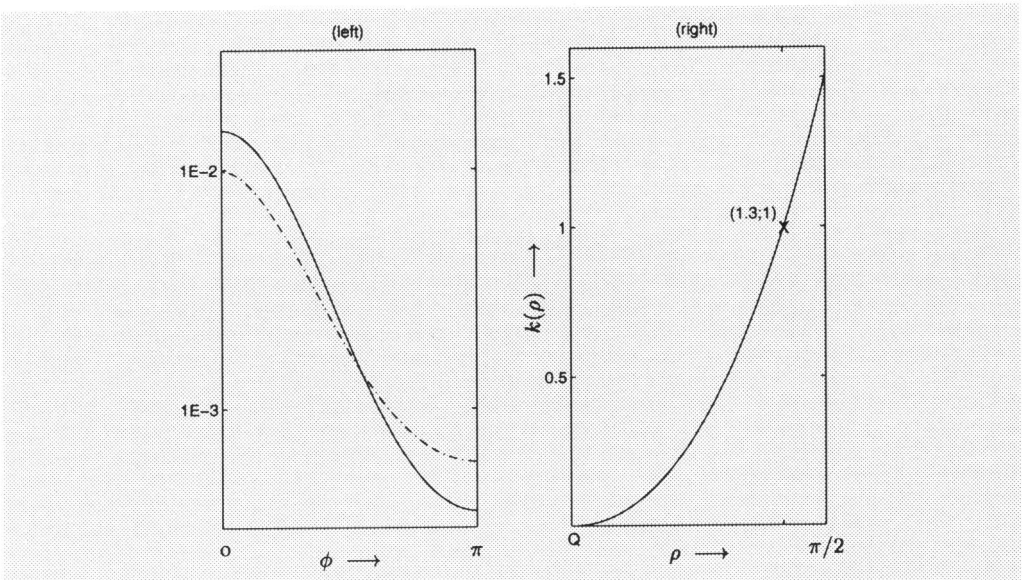

FIGURE 9.3: *Error* $|R(z) - \mathrm{e}^z|$ *(solid) and error estimator* $|R(z) - \hat{R}^{(1)}(z)|$ *(dash-dotted) for* $\hat{b}_0 = 0.02$ *on upper part of* $\partial\mathcal{A}(\pi/2) = \{z = \pi \mathrm{e}^{\mathrm{i}\varphi}/2 \; ; \; 0 \leq \varphi \leq \pi\}$ *(left), and the maximum ratio* $k(\rho) = \max |(R(z) - \mathrm{e}^z)/(R(z) - \hat{R}^{(1)}(z))|$ *for* $|z| \leq \rho$ *and* $0 \leq \rho \leq \pi/2$ *(right).*

Let us now consider linear constant coefficient systems

$$y' = Jy$$

solved with the method pair $(R, \hat{R}^{(1)})$. Because the error estimator is a rational function analytic in $\mathcal{A}(\rho)$, it follows from the spectral theorem, and the maximum modulus theorem, that the estimated relative error in the system is bounded by

$$\|R(hJ) - \hat{R}^{(1)}(hJ)\|_2 \leq \max_{\partial\mathcal{A}(\rho)} |R(z) - \hat{R}^{(1)}(z)| =: \Delta(\rho)$$

for all matrices J with $\|hJ\|_2 \leq \rho$. Since the error estimator has local order 4, we have $\Delta(\rho) = \mathcal{O}(\rho^4)$. From Figure 9.4 we see that the estimated error exceeds the actual error on $\mathcal{A}(1.3)$, and in the following we may therefore take $0 < \rho < 1.3$. By formally approximating the matrix exponential e^{hJ} by a polynomial $P_{\exp}(hJ)$ such that $\|\mathrm{e}^{hJ} - P_{\exp}(hJ)\|_2 \leq \delta$ on $\mathcal{A}(\rho)$, it follows that the actual relative error in the system is bounded by

$$\begin{aligned} \|R(hJ) - \mathrm{e}^{hJ}\|_2 &\leq \|R(hJ) - P_{\exp}(hJ)\|_2 + \delta \\ &\leq \max_{|z| \leq \rho} |R(z) - P_{\exp}(z)| + \delta \end{aligned}$$

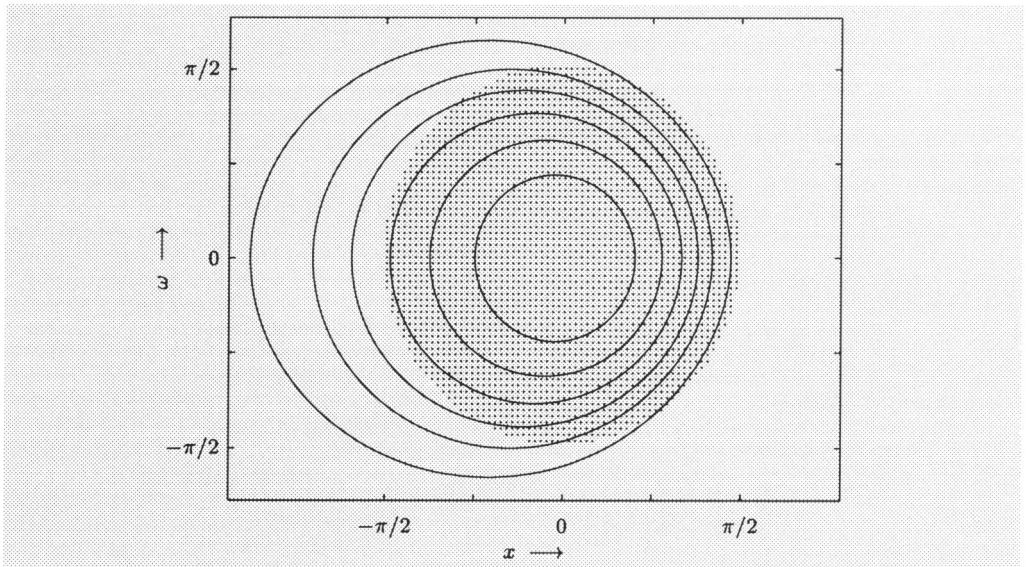

FIGURE 9.4: *Level contours of* $|R(z) - e^z|/|R(z) - \hat{R}^{(1)}(z)| = \kappa$ *for* $\kappa = 0.2(0.2)1.2$. *The shaded area is* $\mathcal{A}(\pi/2)$. *The plot was obtained for* $\hat{b}_0 = 0.02$.

$$\begin{aligned} &\leq \max_{|z|\leq\rho} |R(z) - e^z| + 2\delta \\ &\leq \max_{|z|\leq\rho} |R(z) - \hat{R}^{(1)}(z)| + 2\delta \\ &= \Delta(\rho) + 2\delta \end{aligned}$$

for all J with $\|hJ\|_2 \leq \rho$. Note that δ can be made arbitrarily small. Thus we have a bound on the actual error in linear systems, in terms of the error estimator, uniform with respect to the conditioning of J.

It is also of interest to bound the actual error directly in terms of the estimated error, i.e. we would like to find a constant $C(\rho) < 1$ such that for all vectors y,

$$\|(R(hJ) - e^{hJ})y\|_2 \leq C(\rho)\|(R(hJ) - \hat{R}^{(1)}(hJ))y\|_2. \tag{9.21}$$

This can be obtained in a similar manner. By (9.18) and (9.19),

$$K(z) = \frac{R(z) - e^z}{R(z) - \hat{R}^{(1)}(z)} = \frac{(\gamma z - 1)(P_R(z) - Q_R(z)e^z)}{\hat{b}_0 z^4}.$$

Note that $P_R(z) - Q_R(z)e^z = Q_R(z)\mathcal{O}(z^6)$, hence

$$K(z) = \frac{(\gamma z - 1)Q_R(z)}{\hat{b}_0}\mathcal{O}(z^2)$$

because of the pole–zero cancellation at the origin. Thus $K(z)$ is regular in $\mathcal{A}(\rho)$ with a double zero at the origin; this is also clearly seen in Figure 9.3 (right). It follows from (9.21) by the pole–zero cancellation that

$$C(\rho) = \sup_{\|hJ\|_2 \leq \rho} \|K(hJ)\|_2.$$

Now, in order to apply the spectral theorem, we again approximate e^{hJ} by $P_{\exp}(hJ)$ and consider instead

$$\tilde{K}(z) = \frac{(\gamma z - 1)(P_R(z) - Q_R(z)P_{\exp}(z))}{\hat{b}_0 z^4}.$$

By taking the degree of $P_{\exp}(z)$ suitably high, we have $\|\tilde{K}(hJ) - K(hJ)\|_2 \leq \delta$, therefore

$$\begin{aligned} \|K(hJ)\|_2 &\leq \|\tilde{K}(hJ)\|_2 + \delta \\ &\leq \max_{|z|\leq\rho} |\tilde{K}(z)| + \delta \\ &\leq \max_{|z|\leq\rho} |K(z)| + 2\delta \\ &= k(\rho) + 2\delta \end{aligned}$$

for all J with $\|hJ\|_2 \leq \rho$. Thus, the actual error is never underestimated on $\mathcal{A}(1.3)$ for linear constant coefficient systems.

We finally remark that the latter result depends on the pole–zero cancellation at the origin. This implies that the result is not valid for more general classes of problems. This comes as no surprise, however, as the error estimator does not contain the same elementary differentials as the actual error; it is therefore not possible to prove that the error estimator is an upper bound for the error in general nonlinear problems.

9.6 Concluding remarks

Since the RADAU5 code uses $\hat{b}_0 = \gamma$, [HW96a], our new estimator with $\hat{b}_0 = 0.02$ has approximately 14 times smaller magnitude without significant underestimation of the error. This leads to approximately 70% larger steps, a better agreement between requested and achieved accuracy, and, for a given tolerance, improved performance. There are in principle two different ways of testing a method's performance: either one investigates achieved precision vs. work or one investigates achieved precision vs. requested precision. The new value of $\hat{b}_0$ affects only the latter. As the method's achieved precision for a given amount of work is the same, it might be argued that changing $\hat{b}_0$ is equivalent to rescaling the tolerance. However, a code which for a given tolerance consistently overestimates the error will from a user's perspective appear to be 'slow'; there is no indication of what a proper rescaling of the tolerance might be. We argue that the tolerance parameter ought to have some 'absolute' meaning in different codes that aim for controlling local errors. This requires and motivates the analysis presented in this paper and is in agreement with the ultimate goal of developing high–order methods: to achieve high accuracy using

large steps. It is always preferable—from a practical as well as theoretical perspective—that the estimated errors are of the right order of magnitude without invoking tolerance rescaling.

The design process above has also been used in PSIDE (see Chapter 11). This code is based on the 4–stage Radau IIA method, for which we obtained $\hat{b}_0 = 0.01$. Practical experience with these error estimators is affirmative; see §12.6 or [LSV96] for numerical experiments with PSIDE.

Chapter 10

Application: Computation of elliptic Fekete point sets

Abstract The objective of this chapter is threefold. Firstly, we want to compute elliptic Fekete point sets. This problem is of obvious interst in many areas of scientific modeling and is normally viewed as a global optimization problem. Secondly, a methodology will be presented to model the problem by a set of Differential–Algebraic equations (DAEs) and finally, we wish to compare the performance of PSIDE on the set of DAEs with that of an Lipschitz Global Optimatization (LGO) package applied to the original formulation of the problem. As a reference, we include the results of the DAE code RADAU5 by Hairer & Wanner as well. Fekete point configurations with up to 150 points are considered, which give rise to large computational efforts.

10.1 Introduction

We shall consider the following classical problem: given the unit sphere (ball) B in the Euclidean real space $\mathbb{R}^3$, and a positive integer n, find the n-tuple of points (unit length vectors)

$$x(n) = \{x_i,\ i = 1, \ldots, n\}\ , \qquad x_i = (x_{i1}, x_{i2}, x_{i3})$$

on the surface S^2 of B, which maximizes the product of distances between all possible pairs $\{x_i, x_j\}$, $1 \leq i < j \leq n$. In other words, we are interested in finding the global maximum of the function

$$f_n(x(n)) = \prod_{1 \leq i < j \leq n} \|x_i - x_j\|\ , \qquad x_i \in S^2\ , \tag{10.1}$$

where $\|\cdot\|$ indicates the Euclidean norm. A set of vectors $x^*(n) = \{x_i^*,\ i = 1, \ldots, n\}$, where $x_i^* \in S^2$, which satisfies the relations

$$f_n^* = f_n(x^*(n)) = \max_{x(n)} f_n(x(n))\ , \qquad x_i \in S^2,\ \ (i = 1, \ldots, n)\ , \tag{10.2}$$

is called the set of elliptic Fekete points of order n [Fek23]. We shall name (10.2) the Fekete (global optimization) problem.

Let us note first of all that—by the classical theorem of Weierstrass—the optimization problem (10.2) has globally optimal solution(s). Second, although—for obvious reasons of symmetry—there are infinitely many vector sets $x^*(n)$ which satisfy (10.2), the solution can easily be made unambiguous (as will be seen in §10.3). Consequently, we shall analyze the problem of finding $x^*(n)$, and the corresponding function value $f_n^* := f_n(x^*(n))$.

The analysis and determination of elliptic Fekete point sets has been of great theoretical interest for several decades: consult, e.g., [Fek23, SS93]. Apparently, it also represents a longstanding numerical challenge: Pardalos [Par95] states it as an open problem. Additionally, because of the direct relation of the formulation (10.2) to models in potential theory [Tsu59], the solution of the Fekete problem (and its possible modifications) has also important practical aspects; we shall return to this point later.

We will start with a short overview of some analytical results concerning Fekete points and related topics, followed by a description of the chosen parameterization of Fekete point sets. In §10.4 and §10.5 the Lipschitzian Global Optimization (LGO) approach and the formulation in terms of Differential-Algebraic Equations (DAEs) will be discussed, respectively. We give a summary of the numerical results and the corresponding performances of both approaches in §10.6. The last section presents some concluding remarks and future perspectives.

10.2 A brief review of some analytical background

The following notes are largely based on the works of Tsuji, Shub and Smale mentioned above.

Let D be a bounded closed set in $\mathbb{R}^3$ which contains infinitely many points. Taking n vectors $z_1, ..., z_n$ from D, define (cf. (10.1)) $z(n) = \{z_1, \ldots, z_n\}$,

$$V_n(z(n)) := \prod_{1 \le i < j \le n} \|z_i - z_j\| \,, \tag{10.3}$$

and

$$V_n^* := V_n(z^*(n)) := \max_{z(n)} V_n(z(n)) \,. \tag{10.4}$$

Define now the normalized value of V_n^* by

$$d_n := d_n(D) \;\; := \sqrt[\binom{n}{2}]{V_n^*} \; > \; 0 \;; \tag{10.5}$$

then the following general result—due to Fekete (1923)—is valid.

THEOREM 10.1 *$d_{n+1} \le d_n$; therefore $\tau(D) = \lim_{n \to \infty} d_n$ exists.*

PROOF See Tsuji (1959), p. 71. □

DEFINITION 10.1 *The quantity $\tau(D)$ is called the transfinite diameter of the set D.*

The apparent connection of Fekete's transfinite diameter with certain problems of packing—i.e., 'find a set of points in D which are located so that no two are very close together'—is discussed, e.g., by Lubotzky, Phillips, and Sarnak [LPS86]. In this context, they also refer briefly to the connection of the transfinite diameter and the so-called elliptic capacity. In problems of finding electrostatic equilibria, the resulting point configurations—modeling repellent bodies—are located on a corresponding equipotential surface. Obviously, physically stable, minimal energy configurations are of great importance also in other areas of natural sciences, most notably, in physics and chemistry. Although both the topology of the potential surface in question and the functional form (the underlying analytical description) of characterizing the 'goodness' of point configurations may vary, the result described by Theorem 10.1 bears direct relevance to such problems, under very general conditions.

Shub and Smale [SS93, p. 9] remark that the transfinite diameter of the sphere of radius $\frac{1}{2}$ equals $\mathrm{e}^{-\frac{1}{2}}$. This directly leads to the estimate (recall (10.2))

$$d_n(S^2) \;=\; \sqrt[\binom{n}{2}]{f_n^*} \approx 2\mathrm{e}^{-\frac{1}{2}} = 1.21306132\ldots\,, \tag{10.6}$$

the approximation is valid for sufficiently large n. Theorem 10.1 immediately provides also a lower bound for the solution of the maximization problem in (10.2):

$$f_n^* \geq \left(2\mathrm{e}^{-\frac{1}{2}}\right)^{\binom{n}{2}}\,. \tag{10.7}$$

This estimate shows the rate of increase of the global optimum value, as a function of the number of Fekete points in the optimal configuration. One can also use the estimate $d_{n+1} \leq d_n$, which directly leads to

$$f_{n+1}^* \leq (f_n^*)^{\frac{n+1}{n-1}}\,. \tag{10.8}$$

The pair of relations (10.7)–(10.8) provides valid lower and upper bounds; (10.8) also bounds the rate of increase of subsequent optimal function values in the Fekete problem.

Concluding this brief review of some essential analytical background, let us note finally that Shub and Smale also refer to the apparently significant numerical difficulty of finding the globally optimal configuration $x^*(n)$, for a given—not too small—n. Difficulties arise due to several reasons: viz., the above mentioned various symmetries of the function f_n, and—more essentially—its inherent multiextremality. Obviously, $f_n(x(n))$ equals zero, whenever (at least) two points x_i coincide. Furthermore, see (10.7), its maximal value very rapidly increases as a function of n. These properties together lead to functions f_n which tend to change in an extremely 'abrupt' manner, making any perceivable numerical solution procedure inherently tedious.

In the following two sections, first we shall introduce a suitable problem representation, and then consider a global optimization approach to solving Fekete problems (approximately), in a robust and numerically viable sense.

10.3 Unique parametric representation of n-tuple point configurations on S^2

It is a natural approach to represent arbitrary point configurations on the surface, S^2, by introducing spherical coordinates. Let us denote the three unit vectors in the usual Cartesian coordinate setting by e_1, e_2, and e_3. Furthermore, for $x_i \in S^2$, let β_i denote the angle between x_i and its projection onto the plane defined by e_1 and e_2; and α_i denote the angle between this projection and e_1. Then the n-tuple $x(n)$ —consisting of corresponding unit length vectors x_i, $i = 1, \ldots, n$—is described by

$$\begin{aligned} x_{i1} &= \cos(\alpha_i)\cos(\beta_i)\,, \\ x_{i2} &= \sin(\alpha_i)\cos(\beta_i)\,, \qquad \begin{pmatrix} 0 \le & \alpha_i & < 2\pi \\ -\pi/2 \le & \beta_i & \le \pi/2 \end{pmatrix} \\ x_{i3} &= \sin(\beta_i)\,. \end{aligned} \tag{10.9}$$

We shall also use the equivalent parametrization, with the auxiliary variables ξ_i

$$\begin{aligned} 0 \le \;& \alpha_i \;< 2\pi\,, \\ -1 \le \;& \xi_i \;\le 1\,; \qquad (-\pi/2 \le \beta_i := \arcsin(\xi_i) \le \pi/2)\,. \end{aligned} \tag{10.10}$$

This results in replacing the calculation of x_{i3} in (10.9) simply by $x_{i3} = \xi_i$. The reparameterization has the advantage that if α_i and ξ_i are taken from a uniform distribution from their domains, then the corresponding points x_i have a uniform distribution on the sphere. This is important for a random search as it is used throughout the global search phase of LGO.

In order to eliminate rotational symmetries, one can select and fix three angles in the spherical representation (10.9) of $x(n)$. We choose

$$\alpha_1 = \beta_1 = \beta_2 = 0 \qquad (\text{i.e., } \alpha_1 = \xi_1 = \xi_2 = 0)\,. \tag{10.11}$$

Geometrically, this means that the unit vector $e_1 = (1, 0, 0)$ is always a component of the optimized Fekete point configuration. Additionally, at least another (the second) vector in the Fekete set sought belongs to the $\{e_1, e_2\}$-plane. This convention effectively reduces the number of unknown parameters in $x(n)$ to $2n - 3$.

10.4 Applying LGO approach

Since S^2 is bounded and closed, and the objective function $f_n(x(n))$ in (10.2) is continuously differentiable, it is also Lipschitz-continuous on $S^2 \times S^2 \times \cdots \times S^2 = \left(S^2\right)^n$. In other words, for any given n and corresponding f_n, there exists a Lipschitz-constant $L = L(n)$ such that for all possible pairs $x(n)$, $\widetilde{x}(n)$ from $\left(S^2\right)^n$ we have

$$|f_n(x(n)) - f_n(\widetilde{x}(n))| \le L\|x(n) - \widetilde{x}(n)\|_\Sigma\,. \tag{10.12}$$

The norm $\|x(n) - \widetilde{x}(n)\|_\Sigma$, defined on $\left(S^2\right)^n$, is a sum of Euclidean norms.

As mentioned earlier, the function f_n is expected to become very 'steep' in certain neighborhoods in $\left(S^2\right)^n$, especially when n becomes large. The complicated structure of function f_n can also be simply visualized, observing that the derivative of f_n has a non-polynomially increasing number of zeros—as a function of n—indicating local minima, maxima and saddle points. Consequently, we shall consider the Fekete problem (10.2) as an instance from the broad category of Lipschitz global optimization problems, without further—more narrow, and algorithmically exploitable—specification. Note additionally that only simple lower and upper bound ('box') constraints are explicitly stated by the parametrization (10.9)–(10.10).

The underlying global convergence theory of Lipschitz optimization algorithms is discussed in detail by Pintér [Pin96], with numerous references therein. This monograph also presents details on implementing algorithms for continuous and Lipschitz global optimization, and reviews a number of prospective applications and case studies.

The numerical results obtained on the basis of a program system called LGO—abbreviating Lipschitz Global Optimization—are given in §10.6 and compared with the results obtained via an alternative approach which will be described in the next section. For more details on LGO, consult [Pin97].

10.5 Formulation for DAE approach

As already mentioned, we have used two approaches to approximate Fekete point sets numerically. The previous section dealt briefly with a global optimization approach. Another way to approximate Fekete point sets is based upon the numerical solution of an index 2 system of differential-algebraic equations (DAEs). For more details on DAEs see [BCP89, HW96b]. This section starts with a derivation of the DAE formulation. We will show that the stable steady states of these DAEs coincide with the optima of the function f_n in (10.1). Some practical remarks concerning the numerical implementation of this approach are also highlighted.

Let us consider a set of n repellent particles on the unit sphere. The coordinates of the i^{th} particle are denoted by x_i. Due to the dynamic behavior of the particles, these coordinates will be parametrized by a time variable, t. The particles are restricted in such a way that they will stay on surface of the the unit sphere in $\mathbb{R}^3$; $x_i(t) \in S^2$. We define the repulsive force on particle i caused by particle j by

$$F_{ij} = \frac{x_i - x_j}{\|x_i - x_j\|^\gamma} \,. \tag{10.13}$$

Note that the choice $\gamma = 3$ can be interpreted as an electrical force affecting particles with unit charge. Furthermore, we imply an adhesion force on the particles, due to which the particles will stop moving after some time. Denoting the configuration of the particles at time t by $x(t) = \{x_1(t), \ldots, x_n(t)\}$, Lagrangian mechanics tells us that $x(t)$ satisfies the following system of differential-algebraic equations:

$$x' \;=\; q, \tag{10.14}$$

$$q' = g(x,q) + G^{\mathrm{T}}(x)\lambda, \tag{10.15}$$
$$0 = \phi(x), \tag{10.16}$$

where q is the velocity vector, $G = \partial\phi/\partial x$ and $\lambda \in \mathbb{R}^n$. The function $\phi : \mathbb{R}^{3n} \to \mathbb{R}^n$ is the constraint, which states that the particles cannot leave the unit sphere:

$$\phi_i(x) = x_{i,1}^2 + x_{i,2}^2 + x_{i,3}^2 - 1.$$

The function $g : \mathbb{R}^{6n} \to \mathbb{R}^{3n}$ is given by $g = (g_i)$, $i = 1, \ldots, n$, where

$$g_i(x,q) = \sum_{j \neq i} F_{ij}(x) + A_i(q),$$

where F_{ij} is given by (10.13). The function A_i is the adhesion force affecting particle i and is given by the formula

$$A_i = -\kappa q_i.$$

Here, κ is set to 0.5. Without this adhesion force, the particles would not stop moving, because the system would preserve its energy. The term $G^{\mathrm{T}}(x)\lambda$ in (10.15) represents the normal force which keeps the particles on S^2.

Let us denote the final configuration by $\widehat{x} = \{\widehat{x}_i, i = 1, \ldots, n\}$. Since we know that the speed of this final configuration is 0, we can substitute $q = 0$ and $x = \widehat{x}$ in formula (10.15), thus arriving at

$$0 = \sum_{j \neq i} F_{ij}(\widehat{x}) + G^{\mathrm{T}}(\widehat{x})\lambda \ ,$$

which is equal to

$$\sum_{i \neq j} \frac{\widehat{x}_i - \widehat{x}_j}{\|\widehat{x}_i - \widehat{x}_j\|^{\gamma}} = -2\lambda_i \widehat{x}_i \, . \tag{10.17}$$

If we, on the other hand, take the logarithm (which is a monotonous function) of $f_n(x(n))$ in (10.1) and differentiate $\log(f_n(x(n)))$ with respect to x_i, then, by applying the method of Lagrange multipliers, we know that f_n has a (local) maximum at x, where x satisfies

$$\nabla_i \log(f_n(x)) = \sum_{i \neq j} \frac{x_i - x_j}{\|x_i - x_j\|^2} = \zeta_i x_i \, . \tag{10.18}$$

Here, ζ_i is the Lagrange multiplier. Comparing (10.18) and (10.17) tells us that computing $\widehat{x}$ for $\gamma = 2$ gives the (local) optima of the function f_n. In principle by solving the system (10.14) - (10.16) it is possible to arrive at the global maximum by varying the initial values and the adhesion parameter κ. However, numerical experiments show that for $n \leq 150$, even with a constant κ and a fixed strategy for choosing the initial

values, one obtains values for f_n that satisfy the conditions (10.7)–(10.8) and are at least as large as those obtained by the LGO implementation. (This will be shown in §10.6.)

Now we describe how the DAE system given by the equations (10.14) - (10.16) and $\gamma = 2$ can be solved numerically. Since (10.16) is a position constraint, the system is of index 3. To arrive at a more stable formulation of the problem, we stabilize the constraint (see Brenan et al. (1989), p. 153) by replacing (10.14) by

$$x' = q + G^{\mathrm{T}}(p)\mu, \tag{10.19}$$

where $\mu \in \mathbb{R}^n$, and appending the differentiated constraint

$$0 = G(x)q. \tag{10.20}$$

The system (10.19), (10.15), (10.16), (10.20) is now of index 2; the variables x and q are of index 1, the variables λ and μ of index 2.

We choose the initial positions $x_i(0)$ on the intersection of S^2 and the $\{e_1, e_2\}$-plane, except the first particle, which is initially in $(0, 0, 1)$. Choosing $q(0) = 0$ yields $\mu(0) = 0$ and $\phi_i'(0) = \langle 2x_i(0), q_i(0)\rangle = 0$. Consequently,

$$\begin{aligned} \phi_i''(0) &= \langle 2x_i(0), q_i'(0)\rangle \\ &= \langle 2x_i(0), g_i(x(0), q(0)) + 2\lambda_i(0)x_i(0)\rangle. \end{aligned}$$

Requiring $\phi_i''(0) = 0$ gives

$$\lambda_i(0) = -\frac{\langle x_i(0), g_i(x(0), q(0))\rangle}{2\langle x_i(0), x_i(0)\rangle} = -\frac{1}{2}\langle x_i(0), g_i(p(0), q(0))\rangle.$$

The problem is now of the form

$$M\frac{\mathrm{d}y}{\mathrm{d}t} = w(y), \quad y(0) = y_0, \tag{10.21}$$

with

$$M = \begin{pmatrix} I_{6n} & 0 \\ 0 & 0 \end{pmatrix},$$

where I_{6n} is the identity matrix of dimension $6n$,

$$y \in \mathbb{R}^{8n}, \quad 0 \le t \le t_{\mathrm{end}}\,, \quad y = \begin{pmatrix} x \\ q \\ \lambda \\ \mu \end{pmatrix} \quad \text{and} \quad w(y) = \begin{pmatrix} q + G^{\mathrm{T}}\mu \\ g + G^{\mathrm{T}}\lambda \\ \phi \\ Gq \end{pmatrix}.$$

Here, t_{end} is chosen such that

$$|q_i(t_{\mathrm{end}})| < 10^{-14}, \qquad \forall\, i \in \{1, 2, \ldots, n\}. \tag{10.22}$$

Numerical experiments show that if $t_{\mathrm{end}} = 1000$, then (10.22) holds for $n \le 150$.

Solving the problem numerically leads to a phenomenon that one might call numerical bifurcation. Assume that two particles x_i and x_j are close to each other at time t_1 with $x_{i,1}(t_1) > x_{j,1}(t_1)$. It may happen that the numerical integration method applied with finite error tolerance τ computes a new stepsize h_τ such that $x_{i,1}(t + h_\tau) > x_{j,1}(t + h_\tau)$, whereas the same method applied with error tolerance $\tilde{\tau}$ results in a stepsize $h_{\tilde{\tau}}$ for which $x_{i,1}(t + h_{\tilde{\tau}}) < x_{j,1}(t + h_{\tilde{\tau}})$. This means that for different error tolerances, the numerical integration method may compute paths of particles that differ significantly. The occurrence of this phenomenon is irrespective of the scale of the error tolerance and can happen for every value of n. Although it is more probable for larger values of n. However, the quantity of interest here is (10.1) which is independent of the path that the particles followed to arrive at the final configuration.

To solve the DAE we use the parallel code PSIDE, specified in Chapter 11. As a reference we include the results of the DAE solver RADAU5 by Hairer and Wanner [HW96a], which is an implementation of the 3-stage implicit Runge–Kutta method of Radau IIA type. For more information related to this code, we refer to [HW96b]. RADAU5 can integrate problems of the form (10.21) up to index 3.

As an example, Figure 10.1 depicts the solution obtained by PSIDE for $n = 20$. The same solution in the $\{\alpha, \beta\}$-plane (cf. (10.9))—after a rotation such that (10.11) is fulfilled—is shown in Figure 10.2.

REMARK For $n = 20$ the DAE formulation of the Fekete problem is included in the Test Set for IVP Solvers [LSV96]. ◇

10.6 Numerical results and discussion

From the previous exposition it should be clear that the numerical determination of Fekete point sets leads to rapidly growing computational demands which can easily become prohibitive. Therefore—although 'precise' globally optimal solutions have been sought—the results reported in this section should be considered as numerical approximations obtained with a reasonable computational effort, for the purposes of this exploratory study. The individual solution times on a SGI workstation, Indy with a 194 Mhz R10010SC processor, start with a few seconds for both approaches for small number of points and lead to CPU times between 2 and 17 hours for n in the range of 100 to 150 Fekete points. Even a powerful personal computer is too slow for such a task and memory limitations will become a serious drawback for the DAE approach in case of increasing n. To give an impression: The highest order term of the storage required by RADAU5 and PSIDE is $4(8n)^2$ and $6(8n)^2$ real numbers, respectively. This means that using double precision, these codes need about $2 \cdot 10^3 n^2$ and $3 \cdot 10^3 n^2$ bytes of memory. For $n = 150$ this is about 45 MByte for RADAU5 and 60 MByte for PSIDE, which can be a severe restriction on small computer systems. For the same number of Fekete points, the LGO approach needs only 0.1 MByte. Concerning this comparison of the sizes—especially for

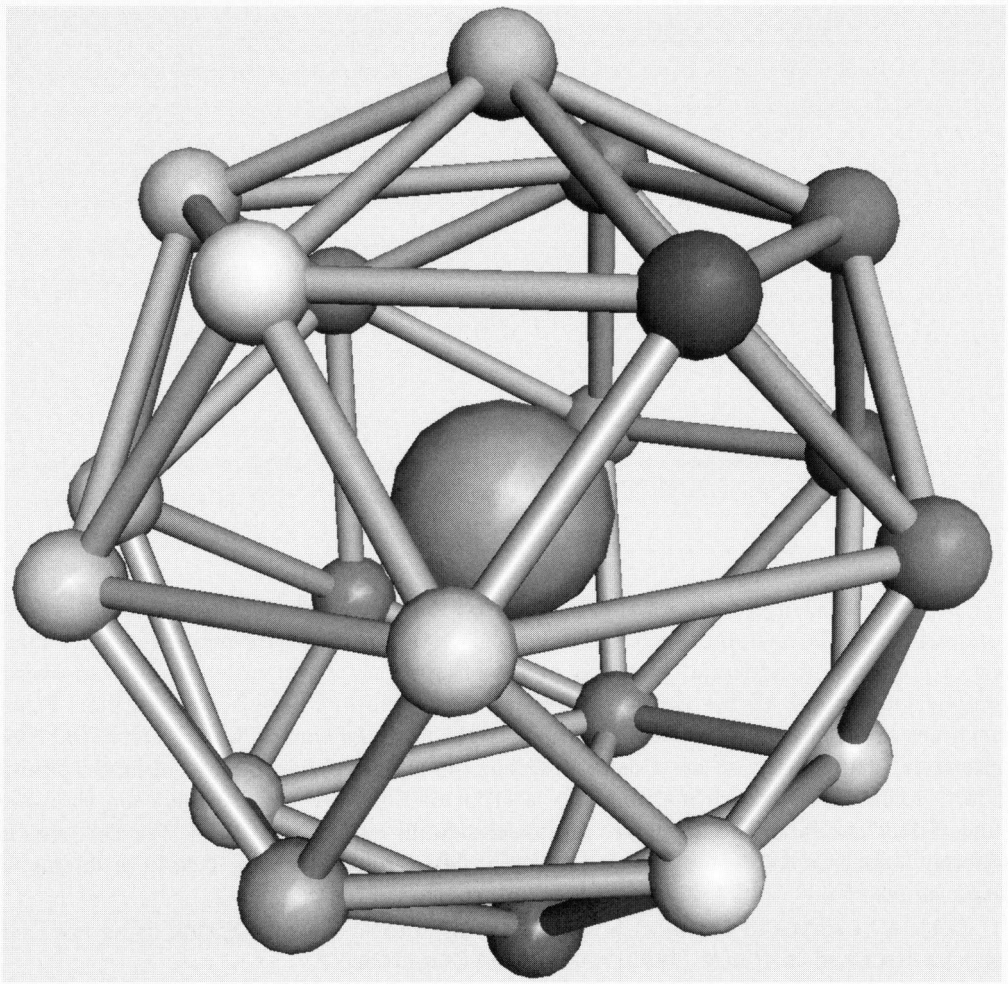

FIGURE 10.1: *Final configuration obtained with PSIDE for $n = 20$. The large ball is centered at the origin and only added to facilitate the 3-D perception.*

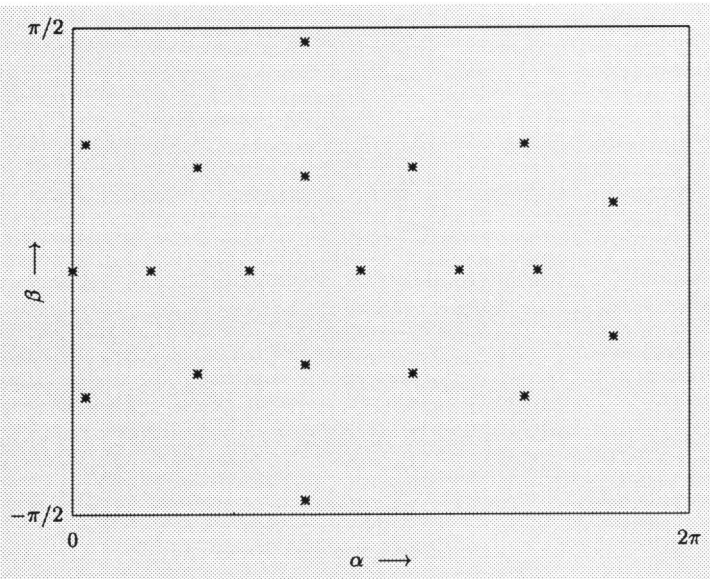

FIGURE 10.2: *Final configuration, as in Figure 10.1, where the Fekete points are given in the* $\{\alpha, \beta\}$*-plane. A rotation has been applied such that (10.11) is fulfilled.*

$n \geq 50$—the LGO approach is favorite. Later on in this section we show a more thorough comparison of the two approaches. Numerical tests can be performed for smaller number of points on a personal computer or a workstation, but in order to give an overall comparison we did all the computations on the above mentioned workstation. Faster machines are useful—and are even available right now—of course, but the essential computational complexity of the Fekete problem remains exponential. Applying a similar global (exhaustive) search methodology to that of LGO, even on a (say) ten thousand times faster machine, the hardware limitations could be easily reached. For this reason, different heuristic solution strategies need to play a role in solving Fekete problems for large values of n.

Table 10.1 serves to summarize the results obtained on a workstation using the LGO version described in Pintér (1995) and the DAE approach.

Concerning the two tables several additional points should be mentioned; see also the notes provided in the table.

1. The CPU timings for PSIDE are done on 1 processor, although it is meant for implementation on a computer with four processors. Using such a machine, we computed that for $n = 20$, the CPU time for PSIDE would be reduced by a factor 3.8. An explanation of how we obtained this speed-up factor is given in [LSV96] or §6.5. For configurations with more than 20 points it is expected that the speed-up will be at

TABLE 10.1: *Summary of the numerical results obtained with LGO, RADAU5 and PSIDE.*

n	$^{10}\log(f^*(n))$[a]			$d(f^*(n))$[b]	CPU in seconds		
	LGO	RADAU5	PSIDE		LGO	RADAU5	PSIDE
3	0.715681[c]	0.715681	0.715681	1.73205	0.32	0.02	0.05
4	1.277905[d]	1.277906	1.277906	1.63299	0.81	0.03	0.09
5	1.919801	1.915913	1.919801	1.55589	1.72	0.06	0.19
6	2.709262[e]	2.709269	2.709269	1.51571	3.11	0.17	0.34
7	3.552441	3.553605	3.553604	1.47645	5.51	0.29	0.52
8	4.528308	4.528830	4.528830	1.45125	8.29	0.49	0.68
9	5.596715	5.597079	5.597079	1.43045	11.24	0.49	0.85
10	6.758096	6.758978	6.758978	1.41318	14.85	0.60	1.15
11	7.998094	7.999912	7.999912	1.39782	22.15	0.83	1.38
12	9.382082	9.383429	9.383429	1.38730	29.05	1.08	1.83
13	10.796868	10.799480	10.799480	1.37548	37.04	1.44	2.52
14	12.330099	12.337356	12.337356	1.36639	46.61	1.68	3.43
15	13.952383	13.961645	13.961645	1.35821	57.78	2.15	4.06
16	15.679583	15.680702	15.680702	1.35105	70.17	4.67	5.55
17	17.476709	17.490362	17.490369	1.34463	84.72	3.49	5.95
18	19.383523	19.391373	19.391373	1.33887	101.07	4.49	7.29
19	21.358636	21.367241	21.367235	1.33338	119.02	5.06	9.68
20	23.437311	23.456734	23.456735	1.32879	139.12	6.07	11.14
25	35.163852	35.176771	35.176759	1.30995	273.17	16.52	29.12
30	49.091838	49.114039	49.114029	1.29689	476.75	32.42	57.23
35	65.157241	65.227582	65.227582	1.28714	757.61	58.50	106.39
40	83.404060	83.531197	83.531197	1.27965	1012.75	138.31	158.65
45	103.832992	103.993419	103.993420	1.27363	1614.29	169.41	229.76
50	126.399795	126.609262	126.609262	1.26868	2222.95	224.81	360.78
60	178.036972	178.291893	178.291893	1.26104	3850.51	586.50	730.29
70	238.216583	238.547125	238.547125	1.25538	5949.21	1573.90	2052.28
80	306.962219	307.343814	307.351658	1.25101	9102.11	3380.64	3321.87
90	384.406738	384.668442	384.668441	1.24751	11950.35	5511.98	8076.43
100	470.001251	470.493394	470.499694	1.24465	17919.00	8844.01	12628.87
125	721.470520	722.227981	722.230435	1.23934	33587.70	23703.40	30428.18
150	1026.298706	1026.946736	1026.937393	1.23565	59967.91	55152.32	64330.46
∞	∞	∞	∞	1.21306[f]			

[a]For definition see (10.2).
[b]For definition see (10.5). The $f^*(n)$ value of PSIDE has been used.
[c]Exact value: $^{10}\log(3\sqrt{3}) = 0.715681882\ldots$
[d]Exact value: $^{10}\log((8/3)^3) = 1.2779061968\ldots$
[e]Exact value: $^{10}\log(512) = 2.7092699609\ldots$
[f]Recall (10.6).

least 3.8, whereas for less Fekete points, one might argue that the CPU times are too small to justify the use of a parallel computer. Of course LGO and RADAU5 could benefit as well from the use of more processors by developing a special purpose strategy for dividing the work. However, in Chapter 1 we argued that such a form of parallelism over the problem lies outside the scope of this thesis.

2. For the values $n = 2, 3, 4$ and 6, the exact analytical solution is trivial, or can be easily verified; with the exception of $n = 2$, however, all values in the tables resulted from numerical calculations. Consequently, all entries are approximate values, except when stated otherwise.

3. Concerning the LGO approach: since the function value f_n^* grows very rapidly as n increases, the resulting (overall) Lipschitzian problem characteristics are also rapidly becoming less favorable. Therefore the value of $f_n(x(n))$ has been directly optimized only up to $n = 6$. Starting from $n = 7$, optimization using the original objective function form has been replaced, by applying a logarithmic transformation.

4. Concerning the LGO approach: 'exact' (exhaustive) search has been attempted for the 'small' values $n = 3, \ldots, 15$. That is, up to $n = 15$, all entries have been calculated by fully automatic LGO execution in which the stated global and local limits imposed on the allowed search effort did not seem to be restrictive. (In particular, the bound on the number of allowable local search steps has never been attained, indicating that the LGO search was completed by finding a solution 'as precise as possible' under the given LGO parametrization.) In order to avoid very excessive runtimes, in the cases $n = 50, 60, \ldots, 125, 150$ the number of global search function evaluations was—based on the analysis of detailed LGO output listings, but still somewhat arbitrarily—restricted by 250 000 to 750 000. In light of the computational effort in smaller dimensional Fekete problems, such limitations could be a bit 'optimistic', and may have stopped the global search phase somewhat prematurely. Furthermore, the local search effort (limited by 100 000 to 300 000) has also been attained, in several higher dimensional cases. Notwithstanding these numerical limitations, all LGO runs provided 'plausible' results, conforming with the theoretical bounds and asymptotics reviewed in §10.2. The global and local search efforts were also chosen in such a way that their sum was comparable to the CPU time for the DAE approach for $n \geq 50$.

5. Concerning the DAE approach: the input parameters for RADAU5 are, independently from n, `h0=atol=rtol=1d-4`, and for PSIDE `atol=rtol=1d-3`. These settings are such that the accuracies delivered by both codes are more or less the same.

6. For both approaches the machine used: SGI workstation, *Indy* with a 194 Mhz R10010SC processor and 512 MByte memory.

7. Compiler: FORTRAN 77 of SGI with optimization: `f77 -O`.

8. Timing function: `ETIME`

From the table we see that for small problem sizes, PSIDE is on one processor about twice as expensive as RADAU5. This factor reduces to ≈ 1.2 when n grows to 150. For almost all cases the results of PSIDE and RADAU5 are more accurate than that of LGO. Except for the above mentioned computer memory limitations, the DAE approach performs better than the LGO approach (according to their given implementations) in terms of CPU time. It should be mentioned here that this optimization problem is special because it can be rewritten as a set of DAEs, for more general optimization problems the solution can not be obtained with a DAE solver and a more general, e.g. LGO style, solver is indispensable.

10.7 Generalizations and application perspectives

An obvious generalization of the Fekete problem—which immediately falls within the scope of the numerical solution strategy suggested—is its extension to arbitrary dimensionality, and for general compact sets. Let D be a bounded closed set in $\mathbb{R}^d$ $d \geq 2$, which contains infinitely many points. Then (recalling the discussion in §10.2) the generalized Fekete configuration problem consists of finding an n-tuple of points $z(n) = (z_1, ..., z_n)$ such that z_i belongs to D, and the product

$$V_n(z(n)) := \prod_{1 \leq i < j \leq n} \|z_i - z_j\| \tag{10.23}$$

is maximal. As noted earlier, problems of this general class have relevance in diverse areas of scientific modeling.

The higher dimensional case is also of interest in the area of nonlinear regression. From linear regression one obtains an ellipsoidal level set, which can be used as an approximation for the level set of the regression variables in the nonlinear case. Evaluation of the regression criterion at points which are distributed in a regular and uniform way on such an ellipse gives good insight into the nonlinearity of the regression problem; the ellipsoid turns into a 'cashew nut', for example. The uniformly distributed sample points on such an ellipsoidal level set can be obtained by solving the Fekete problem (10.23), where D is the ellipsoidal level set and n the number of sample points. Again, the numerical solution approach—Lipschitzian global optimization or DAE formulation— is directly relevant to analyze and solve such problems. This statement remains true, of course, if the 'simple' objective function type (10.23) is replaced by other suitable (Lipschitzian function) models/formulae expressing the 'quality' of the configurations.

Chapter 11

Specification of PSIDE

Abstract PSIDE is a code for solving implicit differential equations on parallel computers. It is an implementation of the four-stage Radau IIA method. The nonlinear systems are solved by a modified Newton process, in which every Newton iterate itself is computed by an iteration process. This process is constructed such that the four stage values can be computed simultaneously. In this chapter, we describe how PSIDE is set up as a modular system and what control strategies have been chosen.

11.1 Introduction

A powerful method for the numerical solution of the system of implicit differential equations

$$\begin{array}{lll} g(t,y,\dot{y}) = 0, & g,y \in \mathbb{R}^d, & \\ t_0 \le t \le t_{\text{end}}, & y(t_0) = y_0, & \dot{y}(t_0) = \dot{y}_0, \end{array} \tag{11.1}$$

is an implicit Runge–Kutta method (IRK). In the class of IRKs, the Radau IIA methods combine high order, $2s-1$, where s is the number of stages, with the nice property of L-stability. However, implementing this method requires high computational costs. PSIDE (abbreviating Parallel Software for Implicit Differential Equations) is an implementation of the four-stage Radau IIA method, where the stages can be computed in parallel. Section 11.2 describes how this is done.

When implementing an IRK, a lot of decisions have to be made. How to form a prediction for the Newton process, when to refactorize the iteration matrix, when to evaluate the Jacobian, how many Newton iterations should be done, what should be the new stepsize, when to reject a step? The answers to these questions for PSIDE are mainly based on Chapter 9 and [GS97, OS96, Ben96, HW96b, Gus92].

In order to have a clear overview of these control strategies, PSIDE is set up modularly. Section 11.3 shows how several modules build up PSIDE. Section 11.4–11.12 each describe one of these modules in detail.

11.2 Parallelism in PSIDE

For solving (11.1) numerically with the four-stage Radau IIA method, we have to solve $\dot{Y}$ from the nonlinear system

$$G(\mathbf{1} \otimes y_n + h(A \otimes I)\dot{Y}, \dot{Y}) = 0\,. \tag{11.2}$$

Here, $\dot{Y} = (\dot{Y}_1^{\mathrm{T}}, \dot{Y}_2^{\mathrm{T}}, \dot{Y}_3^{\mathrm{T}}, \dot{Y}_4^{\mathrm{T}})^{\mathrm{T}}$ is the so-called stage derivative vector of dimension $4d$, where the $\dot{Y}_i$ contain approximations to the derivative values $\dot{y}(t_n + c_i h)$, the abscissa are in $c = (c_1, c_2, c_3, c_4)^{\mathrm{T}}$, the stepsize is denoted by h, the 4×4 Radau IIA matrix by A, the approximation to $y(t_n)$ by y_n, the Kronecker product by $\otimes$, the vector $(1,1,1,1)^{\mathrm{T}}$ by $\mathbb{1}$, and G stands for the stacked values of g, i.e.

$$G(\mathbb{1} \otimes y_n + h(A \otimes I)\dot{Y}, \dot{Y}) := \begin{pmatrix} g(t_n + c_1 h, y_n + h \sum_j a_{1j} \dot{Y}_j, \dot{Y}_1) \\ \vdots \\ g(t_n + c_4 h, y_n + h \sum_j a_{4j} \dot{Y}_j, \dot{Y}_4) \end{pmatrix} .$$

Here and in the sequel, I is an identity matrix of dimension either 4 or d, but its dimension will always be clear from the context. For Radau IIA, $c_s = 1$. Once we obtained $\dot{Y}$, we compute the stage vector Y and y_{n+1} from

$$Y = \mathbb{1} \otimes y_n + h(A \otimes I)\dot{Y}, \qquad y_{n+1} = (e_s^{\mathrm{T}} \otimes I)Y ,$$

where $e_s = (0,0,0,1)^{\mathrm{T}}$. To solve (11.2) we apply a modified Newton process and apply the Butcher transformation $AT = T\Lambda$, where Λ is a block diagonal matrix containing 2 blocks of dimension 2×2, thus arriving at

Scheme:

$\dot{Y}$ `is given by predictor`
`repeat until convergence`
  $Y \leftarrow \mathbb{1} \otimes y_n + h(A \otimes I)\dot{Y}$
  `solve` $(I \otimes M + h\Lambda \otimes J)\Delta\dot{W} = -(T^{-1} \otimes I)G(Y, \dot{Y})$
  $\dot{Y} \leftarrow \dot{Y} + (T \otimes I)\Delta\dot{W}$
`end`

where $M = \partial g / \partial \dot{y}$ and $J = \partial g / \partial y$, both evaluated at some previous approximations. By $P \leftarrow Q$ we mean that Q is assigned to P.

REMARK It would also have been possible to define $Z = Y - \mathbb{1} \otimes y_n$, apply a modified Newton process to

$$G(\mathbb{1} \otimes y_n + Z, ((hA)^{-1} \otimes I)Z) = 0 ,$$

and iterate on Z. Applying Butcher transformations $A^{-1}T = T\Lambda^{-1}$ would then lead to

Scheme:

Z `is given by predictor`
`repeat until convergence`
  $\dot{Y} \leftarrow ((hA)^{-1} \otimes I)Z$
  `solve` $((h\Lambda)^{-1} \otimes M + I \otimes J)\Delta W = -(T^{-1} \otimes I)G(\mathbb{1} \otimes y_n + Z, \dot{Y})$
  $Z \leftarrow Z + (T \otimes I)\Delta W$
`end`

which is equally expensive as iterating on $\dot{Y}$. We prefer not to use additional quantities Z. However, for the problem class $M\dot{y} = f(y)$, iterating on Z is somewhat cheaper. $\diamond$

To compute $\Delta\dot{W}$ we would need to solve two linear systems of dimension $2d$. Instead of doing this, we solve $\Delta\dot{W}$ by the Parallel Iterative Linear system Solver for Runge–Kutta methods (PILSRK) proposed in Chapter 7. In PILSRK we split the matrix Λ in $L + (\Lambda - L)$, where L has distinct positive eigenvalues, and rewrite the equation as

$$(I \otimes M + hL \otimes J)\Delta\dot{W} = (h(L - \Lambda) \otimes J)\Delta\dot{W} - (T^{-1} \otimes I)G(Y, \dot{Y}) .$$

Now we perform an iteration process according to

$$(I \otimes M + hL \otimes J)\Delta\dot{W}^j = (h(L - \Lambda) \otimes J)\Delta\dot{W}^{j-1} - (T^{-1} \otimes I)G(Y, \dot{Y}) . \tag{11.3}$$

If J is a full matrix, then the computation of $h(L - \Lambda) \otimes J$ can be expensive. Since for most applications M is sparser than J (e.g., for ordinary differential equations, $M = \pm I$), we rewrite (11.3) as

$$\begin{aligned}(I \otimes M + hL \otimes J)\left(\Delta\dot{W}^j - \left((I - L^{-1}\Lambda) \otimes I\right)\Delta\dot{W}^{j-1}\right) = \\ - \left((I - L^{-1}\Lambda) \otimes M\right)\Delta\dot{W}^{j-1} - (T^{-1} \otimes I)G(Y, \dot{Y}) .\end{aligned} \tag{11.4}$$

Applying again Butcher transformations $LS = SD$, where D is a diagonal matrix, and m iterations of type (11.4), leads to

Scheme:

$\dot{Y}$ `is given by predictor`
`repeat until convergence`
  $Y \leftarrow \mathbf{1} \otimes y_n + h(A \otimes I)\dot{Y}$
  $\Delta\dot{V}^0 \leftarrow 0$
  `do` $j = 1, \ldots, m$
    `solve` $(I \otimes M + hD \otimes J)(\Delta\dot{V}^j - (B \otimes I)\Delta\dot{V}^{j-1}) =$
      $-(B \otimes M)\Delta\dot{V}^{j-1} - (Q^{-1} \otimes I)G(Y, \dot{Y})$
  `end`
  $\dot{Y} \leftarrow \dot{Y} + (Q \otimes I)\Delta\dot{V}^m$
`end`

where $B = I - (LS)^{-1}\Lambda S$ and $Q = TS$. We now see that the 4 components of dimension d in $\Delta\dot{V}^k$, each corresponding to one stage, can be computed in parallel.

How to form the matrices T, L and S such that PILSRK converges rapidly, can be found in Chapter 7. However, the appendix of this chapter lists all the matrices that play a role in the derivation above.

11.3 PSIDE as modular scheme

Figure 11.1 shows the modules of PSIDE. If a line connects two modules, then the upper module 'calls' the lower one. Table 11.1 describes what the modules basically do. The next sections describe them in more detail.

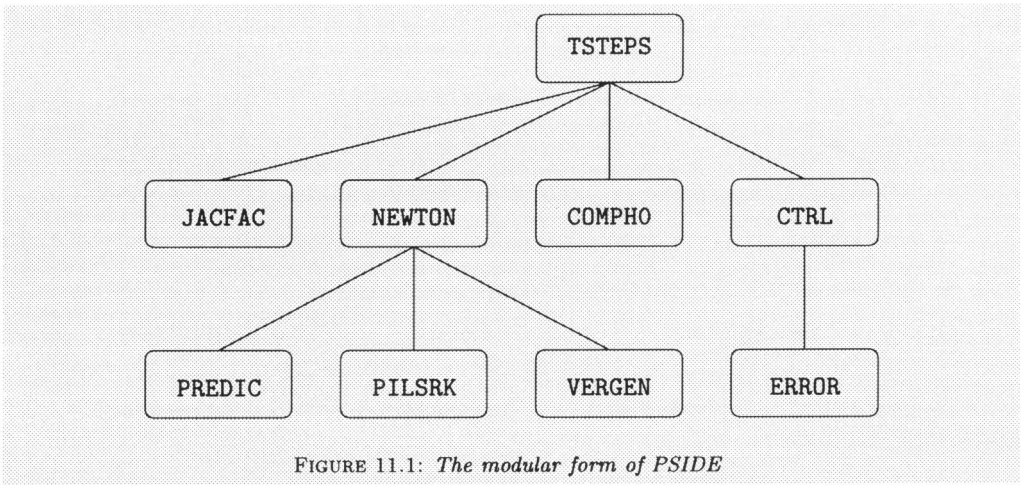

FIGURE 11.1: *The modular form of PSIDE*

Since in an actual implementation, we do not store the sequences $t_0, t_1, \ldots$; $y_0, y_1, \ldots$ and $\dot{y}_0, \dot{y}_1, \ldots$, in the sequel the values t, y and $\dot{y}$ denote the current timepoint, and the approximations to the solution and its derivative at t, respectively.

11.4 Module TSTEPS

This module performs the iteration in time. It uses `COMPHO`, `JACFAC`, `NEWTON`, `CTRL`.

Scheme:

```
compute h by COMPHO
h_LU ← h
t ← t_0
ẏ ← ẏ_0
Ẏ_p ← 1 ⊗ ẏ
first ← true
jacnew ← true
facnew ← true
while ( t < t_end ) do
  depending on jacnew, facnew compute M, J, LU by JACFAC
  compute Y, Ẏ, α, growth, diver, conver by NEWTON
  compute y, ẏ, Ẏ_p, t, h_p, h, h_LU, first, jacnew, facnew by CTRL
end
```

Depending on the boolean variables `jacnew` and `facnew`, module `JACFAC` evaluates the Jacobian matrices M and J and factorizes the iteration matrix $I \otimes M + h_{LU} D \otimes J$. These booleans `jacnew` and `facnew` and the stepsize used for this factorization, h_{LU}, are

TABLE 11.1: *Overview of tasks of modules in Figure 11.1.*

Module	Function
TSTEPS	performs the time-stepping
JACFAC	evaluates the Jacobians and factorizes the iteration matrix
NEWTON	performs the Newton iteration
PREDIC	computes the prediction to start the Newton iteration
COMPHO	computes the initial stepsize
CTRL	decides whether the Jacobians should be updated and the iteration matrix should be factorized
PILSRK	the Parallel Iterative Linear system Solver for Runge–Kutta methods
VERGEN	checks whether the Newton iterates diverge or converge
ERROR	computes the local error estimate

determined in the control module CTRL.

Apart from the vectors $\dot{Y}$ and Y, module NEWTON gives as output the estimated rate of convergence α, and the boolean variables growth, diver and solved. If the growth of the current iterate has a too large increment with respect to y, then growth is true, whereas diver and solved tell that the Newton process is diverging or has converged. Based on α, growth, diver and solved, module CTRL decides whether the current $\dot{Y}$ and Y can be accepted. If so, then it assigns $t \leftarrow t + h$, $y \leftarrow Y_s$, $\dot{y} \leftarrow \dot{Y}_s$ and proposes a new stepsize. The old stepsize and old stage derivative vector are stored in h_p and $\dot{Y}_p$, respectively. If the step is not accepted, then t, y and $\dot{y}$ are not changed and h is recomputed.

11.5 Module JACFAC

Depending on jacnew and facnew, this module evaluates the Jacobians and factorizes the iteration matrix. If the Jacobians are re-evaluated, the boolean jacu2d (Jacobians up to date) is set true.

Variables:

input	jacnew, facnew, h_{LU}
output	jacu2d, M, J, LU

Scheme:

```
if (jacnew) then
  form the Jacobian matrices M, J
  jacu2d ← true
end
if (facnew) then
  factorize the iteration matrix I ⊗ M + h_LU D ⊗ J
end
```

11.6 Module NEWTON

Module NEWTON, which uses the modules PREDIC, PILSRK and VERGEN, performs the Newton iteration.

Variables:

input	$y, \dot{y}, \dot{Y}_p, LU, M, t, h_p, h$
output	$Y, \dot{Y}, \alpha$, growth, diver, slow, solved, exact
local	$\Delta Y, \Delta\dot{Y}, k$, ready

Scheme:

```
ready ← false
k ← 0
compute Ẏ by PREDIC
Y ← 𝟙 ⊗ y + h(A ⊗ I)Ẏ
compute ready, growth by VERGEN
while (not(ready)) do
  k ← k + 1
  compute ΔẎ by PILSRK
  ΔY ← h(A ⊗ I)ΔẎ
  Y ← Y + ΔY
  Ẏ ← Ẏ + ΔẎ
  compute α, growth, diver, slow, solved, exact by VERGEN
end
```

The estimated rate of convergence α and the booleans growth, diver, slow, solved and exact are computed for use in module CTRL.

11.7 Module PREDIC

As starting value for the Newton process we use fourth order extrapolation of the previous stage derivative vector, i.e.

$$\dot{Y} = EY_p,$$

where the $s \times s$ matrix E is determined by the order conditions

$$E(c - \mathbb{1})^k = \left(\tfrac{h}{h_p}c\right)^k, \qquad k = 0, 1, \ldots, s-1\,.$$

(Here, for any vector $a = (a_i)$, the k^{th} power a^k, is understood to be the vector with entries a_i^k.) This means that

$$E = VU^{-1}, \quad U := \left[\mathbb{1} \quad c - \mathbb{1} \quad \cdots \quad (c-\mathbb{1})^{s-1}\right], \quad V := \left[\mathbb{1} \quad \tfrac{h}{h_p}c \quad \cdots \quad \left(\tfrac{h}{h_p}\right)^{s-1}\right].$$

Variables:		
	input	$\dot{Y}_p, h_p, h$
	output	$\dot{Y}$

Scheme: $\dot{Y} \leftarrow EY_p$

We experimented with a lot of other predictors than fourth order extrapolation: Extrapolation of order 3 (using only 3 of the 4 stages in $\dot{Y}_p$), a polynomial of degree 2 that fitted the four stages in Y_p in an—in least squares sense—optimal way, a last step point value predictor (i.e. $\dot{Y} = (\mathbb{1} e_s^{\mathrm{T}} \otimes I)\dot{Y}_p$), and a starting value that is one of these predictors, depending on which predictor yields the smallest residual. We also tried several rational approximations. However, numerous experiments showed that fourth-order extrapolation yields the best overall performance.

11.8 Module COMPH0

This module computes the initial stepsize h_0. It is similar to the strategy used in DASSL [Pet91].

Variables:		
	input	$\dot{y}_0, t_0, t_{\mathrm{end}}$
	output	h_0
	local	$\zeta, h_{\mathrm{def}}, f_h$

Scheme:

$h_0 \leftarrow \mathtt{min}\{h_{\mathrm{def}}, f_h|t_{\mathrm{end}} - t_0|\}$
`if` ($\|\dot{y}_0\|_{\mathrm{scal}} > \zeta/h_0$) `then`
  $h_0 \leftarrow \zeta/\|\dot{y}_0\|_{\mathrm{scal}}$
`end`
$h_0 \leftarrow \mathtt{sign}(h_0, t_{\mathrm{end}} - t_0)$

Parameters:

	value	*source*
ζ	0.5	[Pet91]
h_{def}	10^{-5}	[HW96a]
f_h	10^{-5}	experience

11.9 Module CTRL

This module is a modified version of the one presented in [GS97, Figure 4].

Variables:		
	input	$y, Y, \dot{y}, \dot{Y}, LU, t, t_{\mathrm{end}}, h, h_{LU}, \alpha,$ `first`, `growth`, `diver`, `slow`, `solved`, `jacu2d`, `jacnew`, `exact`
	output	$y, \dot{y}, \dot{Y}_p, t, h_p, h, h_{LU},$ `first`, `jacnew`, `facnew`
	local	$h_r, h_{\mathrm{new}}, h_\alpha, \alpha_{\mathrm{ref}}, \alpha_{\mathrm{jac}}, \alpha_{LU}, f_{\min}, f_{\max}, f_{\mathrm{rig}}, \xi, \omega, n_{\mathrm{rem}}$

Scheme:

$h_\alpha \leftarrow h\alpha_{\text{ref}}/\max\{\alpha, \alpha_{\text{ref}}/f_{\max}\}$
if (solved) then
  compute $y, \dot{y}, \dot{Y}_p, t, h_p, h_r$, first by ERROR
  if ($|t_{\text{end}} - t| > 10u_{\text{round}}|t|$) then
    if jacu2d $\wedge\, \alpha > \alpha_{\text{ref}}$) then
      $h_{\text{new}} \leftarrow \min\{f_{\max}h, \max\{f_{\min}h, \min\{h_r, h_\alpha\}\}\}$
    else
      $h_{\text{new}} \leftarrow \min\{f_{\max}h, \max\{f_{\min}h, h_r\}\}$
    end
    if (not(exact) $\wedge\, \alpha - |h - h_{LU}|/h_{LU} > \alpha_{\text{jac}}$) then
      if (jacu2d) then
        $h_{\text{new}} \leftarrow h/f_{\text{rig}}$
      else
        jacnew $\leftarrow$ true
      end
    end
  end
elseif (growth) then
  $h_{\text{new}} \leftarrow h/f_{\text{rig}}$
elseif (diver) then
  $h_{\text{new}} \leftarrow \min\{f_{\max}h, \max\{f_{\min}h, h_\alpha\}\}$
  jacnew $\leftarrow$ not(jacu2d)
elseif (slow) then
  if (jacu2d) then
    if ($\alpha > \xi\alpha_{\text{ref}}$) then
      $h_{\text{new}} \leftarrow \min\{f_{\max}h, \max\{f_{\min}h, h_\alpha\}\}$
    else
      $h_{\text{new}} \leftarrow h/f_{\text{rig}}$
    end
  else
    $h_{\text{new}} \leftarrow h$; jacnew $\leftarrow$ true
  end
end
$n_{\text{rem}} \leftarrow (t_{\text{end}} - t)/h_{\text{new}}$
if ($n_{\text{rem}} - \lfloor n_{\text{rem}} \rfloor > \omega \vee \lfloor n_{\text{rem}} \rfloor = 0$) then
  $n_{\text{rem}} \leftarrow \lfloor n_{\text{rem}} \rfloor + 1$
else
  $n_{\text{rem}} \leftarrow \lfloor n_{\text{rem}} \rfloor$
end
$h \leftarrow (t_{\text{end}} - t)/n_{\text{rem}}$
facnew $\leftarrow$ jacnew $\vee (|h_{\text{new}} - h_{LU}|/h_{LU} > \alpha_{LU})$
if (facnew) $h_{LU} \leftarrow h$

Parameters:

	value	*source*
α_{ref}	0.25	see below
α_{jac}	0.2	[GS97]
α_{LU}	0.3	see below
$f_{\min}$	0.2	[Ben96, p.52]
$f_{\max}$	2	experience
f_{rig}	2	[Gus92, p.154]
ξ	1.2	experience
ω	0.05	[SSV97]

In this scheme, α_{ref} is the desired rate of convergence. In [GS97] it is shown that, under reasonable assumptions, h_α is the stepsize for which the rate of convergence will be α_{ref}. If the current rate of convergence α is larger than α_{ref}, then h_α will be used for the next Newton process, unless it is greater than h_r, the stepsize proposed by module `ERROR`. [GS97] also derives that, if $\alpha - |h - h_{LU}|/h_{LU} > \alpha_{\text{jac}}$, then the convergence of the Newton process is likely to fail due to an old Jacobian. Such failures are prevented by the strategy above. If nevertheless the Jacobian is fresh, then the assumptions of the theory are not fulfilled and the stepsize is reduced by a rigid factor f_{rig}. The case where `exact` is true, refers to the situation that that (11.2) was solved exactly. This happens e.g. if the function g in (11.1) equals $\dot{y}$.

If `growth` is true (see §11.11), then the stepsize is reduced by a factor f_{rig}, but we do not compute new Jacobians.

For a diverging Newton process, h_α will be the new stepsize. If `slow` is true, then the Newton process is converging too slowly (see §11.11). In this case, the new stepsize is again identified with h_α, if $\alpha > \xi\alpha_{\text{ref}}$. The factor ξ, which has to be > 1, is built in for the case that α is only slightly larger than α_{ref}. Without this factor, the new stepsize for this case would be set equal to h_α, which is only a little bit smaller than the old stepsize, thus leading again to a Newton process that converges too slowly. If $\alpha \leq \xi\alpha_{\text{ref}}$, then the assumptions of the analysis have failed to hold, and the stepsize is rigidly reduced by a factor f_{rig}. For both the diverging and the slowly converging case, the iteration matrix will be factorized, with or without a new Jacobian, depending on `jacnew`.

The formulas of the form $h_{\text{new}} = \min\{f_{\max}h, \max\{f_{\min}h, \cdot\}\}$ prevent the new stepsize to vary from the old stepsize by a factor outside the range $[f_{\min}, f_{\max}]$.

This strategy for updating the Jacobian and refactorizing the iteration matrix is different from most strategies in ODE/DAE software, in the sense that it attempts to adjust the stepsize such that an optimal convergence rate of the Newton process is obtained. Another difference is the strategy for the case that the Jacobian is not updated. Many codes do not change the stepsize in this situation if the relative change of stepsize is in a 'dead-zone' (e.g. RADAU5 uses the dead-zone [1; 1.2]). However, in [Gus92, p.135] it is argued that, in order to arrive at a smooth numerical solution, it is better to remove this strategy 'since a smooth stepsize sequence leads to smoother error control'.

The role of n_{rem} is to adjust the stepsize such that the remaining integration interval

is a multiple of the stepsize. This strategy was taken from [SSV97].

If a new Jacobian is not required, then the iteration matrix will only be factorized in case that the proposed stepsize h_{new} differs significantly from h_{LU}, i.e. ($|h_{\text{new}} - h_{LU}|/h_{LU} > \alpha_{LU}$).

The papers [GS97] and [Gus92] advocate values of α_{ref}, α_{jac} and α_{LU} around 0.2. However, they also suggest to use larger values if the costs of factorizing the iteration matrix are high with respect to the costs of one iteration. Since PSIDE is a parallel code aiming at problems of large dimension, we choose 0.25 for α_{ref} and 0.3 for α_{LU}. Numerous experiments confirmed that these choices yield an efficient code.

11.10 Module PILSRK

This module is the same as presented in §11.2, although an economization is made by computing the first iterate separately. Numerous experiments showed that for index 0 and index 1 problems, performing only 1 inner iteration suffices. On the other hand, [HV97] reveals that for higher-index problems, 2 inner iterations lead to a more robust and efficient behavior.

Variables:

input	$Y, \dot{Y}, LU, M, t, h$
output	$\Delta\dot{Y}$
local	$\Delta\dot{V}, \widetilde{G}, j, m$

Scheme:

$\widetilde{G} \leftarrow (Q^{-1} \otimes I)G(Y, \dot{Y})$
$\Delta\dot{V} \leftarrow -(LU)^{-1}\widetilde{G}$
`if (higher index) then`
  `do` $j = 2, \ldots, m$
    $\Delta\dot{V} \leftarrow (B \otimes I)\Delta\dot{V} - (LU)^{-1}((B \otimes M)\Delta\dot{V} + \widetilde{G})$
  `end`
`end`
$\Delta\dot{Y} \leftarrow (Q \otimes I)\Delta\dot{V}$

Parameters:

	value	*source*
m	2	experience

11.11 Module VERGEN

This module checks the convergence behavior of the Newton process. Most of it is based on [Gus92, §5.2].

Variables:

input	$y, Y, \Delta Y, h, k, \alpha, \alpha_{\text{ref}}$
output	α, `ready`, `growth`, `diver`, `solved`
local	$u_p, u, \tau, \kappa, k_{\max}, \gamma$

Scheme:

```
growth ← false
diver ← false
slow ← false
solved ← false
exact ← false
if ( ∃ i |Y_{i,s}|/max{|y_i|, atol_i} > g_fac ∧ ind_i ≤ 1 ) then
  growth ← true
else
  if ( k = 1 ) then
    u ← ||ΔY||_scal
    α = α_ref
    exact ← u = 0
    solved ← exact
  elseif ( k > 1 ) then
    u_p ← u
    u ← ||ΔY||_scal
    if ( k = 2 ) then
      α ← u/u_p
    else
      α ← α^θ (u/u_p)^{1−θ}
    end
    if ( α ≥ γ ) then
      diver ← true
    elseif ( u α/(1 − α) < τ ∨ u < κ u_round ||y||_scal ) then
      solved ← true
    elseif ( k = k_max ∨ u α^{k_max − k}/(1 − α) > τ ) then
      slow ← true
    end
  end
end
ready ← growth ∨ diver ∨ slow ∨ solved ∨ exact
```

Parameters:

	value	*source*
τ	0.01	see below
κ	100	[Ben96, p.51]
$k_{\max}$	14	see below
γ	1	[Gus92, p.132]
θ	0.5	experience
g_{fac}	100	experience

The boolean variable `growth` monitors whether the current iterate is too large with respect to y. This is necessary to prevent overflow. We use $\max\{|y_i|, \texttt{atol}_i\}$ instead of

$|y_i|$ for the case $y_i = 0$. Experience has shown that it is not efficient to put a limit on the growth of higher-index variables. The variable u is saved for use in the next call of `VERGEN`. The case where `slow` is true, refers to a Newton process that is converging too slowly.

Here and in the sequel, the norm $\|\cdot\|_{\text{scal}}$ is defined by

$$\|X\|_{\text{scal}} = \sqrt{\frac{1}{4d}\sum_{i=0}^{3}\sum_{j=1}^{d}\left(\frac{h^{\texttt{ind}_j-1}X_{id+j}}{\texttt{atol}_j + \texttt{rtol}_j\,|y_j|}\right)^2}, \qquad \text{if} \qquad X \in \mathbb{R}^{4d},$$

and by

$$\|x\|_{\text{scal}} = \sqrt{\frac{1}{d}\sum_{j=1}^{d}\left(\frac{h^{\texttt{ind}_j-1}x_j}{\texttt{atol}_j + \texttt{rtol}_j\,|y_j|}\right)^2}, \qquad \text{if} \qquad x \in \mathbb{R}^{d}.$$

In these formulas, $\texttt{atol}_j$ and $\texttt{rtol}_j$ are the user-supplied absolute and relative error tolerance vectors, respectively, and $\texttt{ind}_j$ contains the user-supplied index of component j. For both index 0 and index 1 variables, $\texttt{ind}_j = 1$.

The value of τ is rather small compared to termination criteria in other codes. E.g., in RADAU5 values around 10^{-1} or 10^{-2} were found to be efficient [HW96b, p.121], and DASSL uses 0.33 [BCP89, p.123]. The reason for this is that τ can be seen as the factor by which the iteration error has to be smaller than the error estimate ϵ. In PSIDE, ϵ is of local order 5 (see §11.12), and the steppoint value of local order 8. Consequently, in order not to let the iteration error spoil the accuracy of the steppoint value, τ has to be smaller than 1, and how much smaller than 1 should depend on the difference between the order of the error estimate and that of the method. This may in part explain why RADAU5, where this difference is 2, and DASSL, where it is 1, use larger values for τ.

The reason for the rather large value of $k_{\max}$ ([HW96b, p.121] advocates values of 7 or 10 for RADAU5) is twofold. Firstly, the order of PSIDE is seven, which is higher than that of e.g. the fifth order RADAU5, so that we need more iterations to find the solution of the non-linear system. Secondly, if PILSRK does not find the exact Newton iterate, a few additional iterations might help.

11.12 Module ERROR

11.12.1 The error estimate in PSIDE

The construction of the error estimate is based on Chapter 9. In order not to have confusion between the previous values of y and $\dot{y}$ and the current ones, we use the time index n in the derivation of the error estimate.

To estimate the error, we use an implicit embedded formula of the form

$$\hat{y}_{n+1} = y_n + h(b_0\dot{y}_n + (b^{\mathrm{T}} \otimes I)\dot{Y} + d_s\hat{\dot{y}}_{n+1}), \tag{11.5}$$

where d_s is the lower right element in D. We eliminate $\hat{y}_{n+1}$ by substituting (11.5) in (11.1) yielding

$$g\big(t_{n+1}, \hat{y}_{n+1}, (hd_s)^{-1}(\hat{y}_{n+1} - y_n - h(b_0\dot{y}_n + (b^{\mathrm{T}} \otimes I)\dot{Y}))\big) = 0\,. \tag{11.6}$$

Solving $\hat{y}_{n+1}$ from (11.6) by a modified Newton process, leads to the recursion

$$\begin{aligned} \hat{y}_{n+1}^{j+1} &= \hat{y}_{n+1}^{j} - hd_s(M + h_{LU}d_sJ)^{-1}g\big(t_{n+1}, \hat{y}_{n+1}^{j}, \\ &\quad (hd_s)^{-1}(\hat{y}_{n+1}^{j} - y_n - h(b_0\dot{y}_n + (b^{\mathrm{T}} \otimes I)\dot{Y}))\big)\,. \end{aligned}$$

Now we set $\hat{y}_{n+1}^{(0)} = y_{n+1}$, and consider the first Newton iterate $\hat{y}_{n+1}^{1}$ as a reference formula by itself, which is

$$\begin{aligned} \hat{y}_{n+1}^{1} &= y_{n+1} - hd_s(M + h_{LU}d_sJ)^{-1}g\big(t_{n+1}, y_{n+1}, \\ &\quad (hd_s)^{-1}(\hat{y}_{n+1} - y_n - h(b_0\dot{y}_n + (b^{\mathrm{T}} \otimes I)\dot{Y}))\big)\,. \end{aligned}$$

In this formula, we determine b such that $\hat{y}_{n+1}^{1}$ is of local order $s+1$, i.e., it satisfies

$$C\,b = (1 - b_0, 1/2, 1/3, \ldots, 1/s)^{\mathrm{T}} - d_s\mathbb{1}\,, \tag{11.7}$$

where $C = (c_{ij})$; $c_{ij} = c_j^{i-1}$. Notice that implying order conditions directly on (11.5) would also lead to (11.7). Chapter 9 describes how to select the parameter b_0 such that the amplitude of the error estimate approximates the true error in y_{n+1}. Carrying out this procedure for the four-stage Radau IIA method yields the value 0.01 for b_0.

We now define the error estimate r by

$$\begin{aligned} r &= \hat{y}_{n+1}^{1} - y_{n+1} \\ &= -hd_s(M + h_{LU}d_sJ)^{-1}g\big(t_{n+1}, y_{n+1}, d_s^{-1}((v^{\mathrm{T}} \otimes I)\dot{Y} - b_0\dot{y}_n)\big)\,, \end{aligned} \tag{11.8}$$

where $v = (v_i)$, with $v_i = a_{si} - b_i$.

We notice that the error estimate (11.8) reduces for ODE problems to the same formula as in RADAU5 [HW96b, p.123, Formula (8.19)]. However, by choosing the reference method as being of the form (11.5), the 'filtering' with the matrix $(M + h_{LU}d_sJ)^{-1}$, needed to 'remove' the stiff error components in the error estimate, arises on purely mathematical grounds.

11.12.2 Stepsize selection

The following module contains the predictive stepsize controller of Gustafsson [Gus92, Listing 5.1] to propose a new stepsize h_r.

Variables:		
	input	$y, Y, \dot{y}, \dot{Y}, LU, t, h_p, h, f_{\max}$, `first`
	output	$y, \dot{y}, \dot{Y}_p, t, h_p, h_r$, `first`, `jacu2d`
	local	$r, h_{\mathrm{rej}}, \epsilon_p, \epsilon_{\mathrm{rej}}, \epsilon, p_{\mathrm{est}}, p_{\min}$, `sucrej`

Scheme:

```
compute r from (11.8)
ε ← ||r||_scal
if ( ε < 1 ) then
  if ( ε = 0 ) then
    h_new ← f_max h
  elseif (first ∨ sucrej) then
    first ← false
    h_r ← ζ h ε^(-1/5)
  else
    h_r ← ζ h^2/h_p (ε_p/ε^2)^(1/5)
  end
  y ← Y_s
  ẏ ← Ẏ_s
  Ẏ_p ← Ẏ
  t ← t + h
  if ( |t_end − t| < 10 u_round)|t|) then t ← t_end
  h_p ← h
  ε_p ← ε
  sucrej ← false
  jacu2d ← false
else
  if (not(first) ∧ sucrej) then
    p_est ← min{5, max{p_min, log(ε/ε_rej) / log(h/h_rej)}}
    h_r ← ζ h ε^(-1/p_est)
  else
    h_r ← ζ h ε^(-1/5)
  end
  h_rej ← h
  ε_rej ← ε
  sucrej ← true
end
```

Parameters:

	value	*source*
ζ	0.8	[Gus92, p.156]
$p_{\min}$	0.1	[Gus92, p.121]

The variables h_{rej}, ϵ_p, ϵ_{rej} and `sucrej` are saved for use in the next call of `ERROR`. Notice that this module is only called if the Newton process has converged. If $\|\epsilon\|_{\text{scal}} < 1$, it steps forward in time, i.e. it updates t, y, $\dot{y}$ and shifts $\dot{Y}_p$, h_p and ϵ_p; the Jacobians are per definition not up to date anymore.

Appendix

In this appendix we provide the method parameters in PSIDE, i.e. the abscissa vector c and the RK matrix A, that define the four-stage Radau IIA method, the matrices D, B, Q and Q^{-1}, defining the Parallel Linear system Solver for Runge–Kutta methods (PILSRK), and the scalar b_0 and the vector v, needed for the embedded reference formula. As additional information, we list the matrices Λ, L, S and T that arose in the derivation of PILSRK.

$$c^{\mathrm{T}} = \begin{pmatrix} 0.08858795951268 & 0.40946686444074 & 0.78765946176085 & 1.00000000000000 \end{pmatrix}$$

$$A = \begin{bmatrix} 0.11299947932312 & -0.04030922072350 & 0.02580237742032 & -0.00990467650726 \\ 0.23438399574737 & 0.20689257393542 & -0.04785712804857 & 0.01604742280653 \\ 0.21668178462322 & 0.40612326386742 & 0.18903651817002 & -0.02418210489982 \\ 0.22046221117674 & 0.38819346884323 & 0.32884431998002 & 0.06250000000001 \end{bmatrix}$$

$$\mathrm{diag}(D) = \begin{bmatrix} 0.15207736897658 & 0.19863166560206 & 0.17370482124555 & 0.22687976652481 \end{bmatrix}$$

$$B = \begin{bmatrix} -3.36398745680207 & -0.44654700754010 & 0 & 0 \\ 25.34203884124225 & 3.36398745680207 & 0 & 0 \\ 0 & 0 & -0.43736727682531 & -0.05805760311840 \\ 0 & 0 & 3.29483348541735 & 0.43736727682531 \end{bmatrix}$$

$$Q = \begin{bmatrix} 2.95257334306175 & 0.31594239005361 & 1.53250361857179 & 0.02760017730665 \\ -7.26634778465530 & -0.87557678542461 & -1.05525925554832 & -0.31127768044595 \\ 3.42024269744602 & 0.94929336342678 & -10.79971906268609 & -2.13491394363799 \\ 34.89702510456449 & 4.37526650476817 & -42.90392657810952 & -5.89600020104167 \end{bmatrix}$$

$$Q^{-1} = \begin{bmatrix} 0.49403714522764 & 0.26941265525930 & -0.20775393051682 & 0.06331582713183 \\ -3.53352093058280 & -2.98586378845007 & 1.75646110158256 & -0.49490947213933 \\ 0.48764145508107 & 0.12393820514650 & 0.04237703393234 & -0.01960507515011 \\ -3.24650638474176 & -1.52301305545687 & -0.23459121597752 & -0.01945253030841 \end{bmatrix}$$

$$b_0 = 0.01$$

$$v^{\mathrm{T}} = \begin{pmatrix} 0.01577537639774 & -0.00973676595201 & 0.00646138955427 & 0.22437976652485 \end{pmatrix}$$

$$\Lambda = \begin{bmatrix} 0.15207736897658 & 0.06790969403105 & 0 & 0 \\ -0.35070903457864 & 0.04202359569373 & 0 & 0 \\ 0 & 0 & 0.17370482124555 & 0.01008488557162 \\ 0 & 0 & -0.40058458777036 & 0.20362278551270 \end{bmatrix}$$

$$L = \begin{bmatrix} 0.15207736897658 & 0 & 0 & 0 \\ -0.35070903457864 & 0.19863166560206 & 0 & 0 \\ 0 & 0 & 0.17370482124555 & 0 \\ 0 & 0 & -0.40058458777036 & 0.22687976652481 \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 7.53333333333333 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 7.53333333333333 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 0.57247400465791 & 0.31594239005361 & 1.32458228286171 & 0.02760017730665 \\ -0.67033600112323 & -0.87557678542461 & 1.28969927047784 & -0.31127768044595 \\ -3.73110064036903 & 0.94929336342678 & 5.28329931272012 & -2.13491394363799 \\ 1.93668410197760 & 4.37526650476817 & 1.51260826973776 & -5.89600020104167 \end{bmatrix}$$

Chapter 12

PSIDE users' guide

Abstract PSIDE – Parallel Software for Implicit Differential Equations – is a code for solving implicit differential equations on shared memory parallel computers. In this paper we describe the user interface.

12.1 Introduction

PSIDE solves Implicit Differential Equations (IDEs) of the form

$$\begin{array}{lll} g(t,y,y') = 0\,, & g, y \in \mathbb{R}^d, & \\ t_0 \le t \le t_{\text{end}}, & y(t_0) = y_0\,, & y'(t_0) = y'_0\,, \end{array} \tag{12.1}$$

were y_0 and y'_0 are such that $g(t_0, y_0, y'_0) = 0$ (for higher-index problems the initial values have to satisfy more conditions; see §12.4). It uses the four-stage Radau IIA method. The nonlinear systems are solved by a modified Newton process, in which every Newton iterate itself is computed by means of the Parallel Iterative Linear system Solver for Runge–Kutta (PILSRK) proposed in Chapter 7. This process is constructed such that the four stage values can be computed simultaneously, thereby making PSIDE suitable for execution on four processors. Full details about the algorithmic choices and the implementation of PSIDE can be found in Chapter 11.

12.2 Subroutine heading of PSIDE

PSIDE is a Fortran 77 routine, whose heading reads

```
      SUBROUTINE PSIDE(NEQN,Y,DY,GEVAL,
     +                 JNUM,NLJ,NUJ,JEVAL,
     +                 MNUM,NLM,NUM,MEVAL,
     +                 T,TEND,RTOL,ATOL,IND,
     +                 LRWORK,RWORK,LIWORK,IWORK,
     +                 RPAR,IPAR,IDID)
      INTEGER NEQN,NLJ,NUJ,NLM,NUM,IND(*),LRWORK,LIWORK,
     +                 IWORK(LIWORK),IPAR(*),IDID
      DOUBLE PRECISION Y(NEQN),DY(NEQN),T,TEND,RTOL(*),ATOL(*),
     +                 RWORK(LRWORK),RPAR(*)
      LOGICAL JNUM,MNUM
      EXTERNAL GEVAL,JEVAL,MEVAL
C
```

```
C      INTENT(IN)     NEQN,JNUM,NLJ,NUJ,MNUM,NLM,NUM,TEND,RTOL,ATOL,IND
C     +               LRWORK,LIWORK
C      INTENT(INOUT)  Y,DY,T,RWORK,IWORK,RPAR,IPAR
C      INTENT(OUT)    IDID
```

The variables listed under INTENT(IN), INTENT(INOUT), and INTENT(OUT) are input, update and output variables, respectively.

12.3 Parameters

NEQN

On entry, this is the dimension d of the IDE (12.1), the number of equations to be solved.

Y(NEQN)

On entry, this array contains the initial value y_0.
On exit, Y contains y(T), the computed solution approximation at T.
(After successful return, T = TEND.)

DY(NEQN)

On entry, this array contains the initial value y'_0.
On exit, DY contains y'(T), the computed derivative approximation at T.
(After successful return, T = TEND.)

GEVAL

This is the subroutine which you provide to define the IDE

```
      SUBROUTINE GEVAL(NEQN,T,Y,DY,G,IERR,RPAR,IPAR)
      INTEGER NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),G(NEQN),RPAR(*)
C     INTENT(IN)     NEQN,T,Y,DY
C     INTENT(INOUT)  IERR,RPAR,IPAR
C     INTENT(OUT)    G
```

For the given values of T, Y, and DY the subroutine should return the residual of the IDE

$$\mathtt{G} = g(\mathtt{T}, \mathtt{Y}, \mathtt{DY}).$$

You must declare the name GEVAL in an external statement in your program that calls PSIDE.

IERR is an integer flag which is always equal to zero on input. Subroutine GEVAL should set IERR = -1 if GEVAL can not be evaluated for the current values of Y and DY. PSIDE will then try to prevent IERR = -1 by using a smaller stepsize.

All other parameters have the same meaning as within subroutine PSIDE.

JNUM
: To solve the IDE it is necessary to use the partial derivatives $J = \partial g/\partial y$. The solution will be more reliable if you provide J via the subroutine JEVAL, in this case set JNUM = .FALSE.. If you do not provide a subroutine to evaluate J, provide a dummy JEVAL, set JNUM = .TRUE. and PSIDE will approximate J by numerical differencing.

NLJ and NUJ
: If J is a full matrix, set NLJ = NEQN, otherwise set NLJ and NUJ equal to the lower bandwidth and upper bandwidth of J, respectively.

JEVAL
: This is the subroutine which you provide to define J (if JNUM .EQ. .FALSE.)

```
      SUBROUTINE JEVAL(LDJ,NEQN,NLJ,NUJ,T,Y,DY,DGDY,RPAR,IPAR)
      INTEGER LDJ,NEQN,NLJ,NUJ,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDY(LDJ,NEQN),RPAR(*)
C     INTENT(IN)    LDJ,NEQN,NLJ,NUJ,T,Y,DY,
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DGDY
```

: For the given values of T, Y, and DY the subroutine should return the partial derivatives, such that

  - DGDY(I,J) contains $\partial g_I(T,Y,DY)/\partial y_J$ if J is a full matrix (NLJ = NEQN);
  - DGDY(I-J+NUJ+1,J) contains $\partial g_I(T,Y,DY)/\partial y_J$ if J is a band matrix ($0 \leq$ NLJ $<$ NEQN) (LAPACK / LINPACK / BLAS storage).

: You must declare the name JEVAL in an external statement in your program that calls PSIDE.

: LDJ denotes the leading dimension of J.

: All other parameters have the same meaning as within subroutine PSIDE.

MNUM
: To solve the IDE it is necessary to use the partial derivatives $M = \partial g/\partial y'$. The solution will be more reliable if you provide M via MEVAL, in this case set MNUM = .FALSE.. If you do not provide a subroutine to evaluate M, provide a dummy MEVAL, set MNUM = .TRUE. and PSIDE will approximate M by numerical differencing.

NLM and NUM
: If M is a full matrix, set NLM = NEQN, otherwise set NLM and NUM equal to the lower bandwidth and upper bandwidth of M, respectively. It is supposed that NLM .LE. NLJ and NUM .LE. NUJ.

MEVAL

This is the subroutine which you provide to define M (if MNUM .EQ. .FALSE.)

```
      SUBROUTINE MEVAL(LDM,NEQN,NLM,NUM,T,Y,DY,DGDDY,RPAR,IPAR)
      INTEGER LDM,NEQN,NLM,NUM,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDDY(LDM,NEQN),RPAR(*)
C     INTENT(IN)    LDM,NEQN,NLM,NUM,T,Y,DY,
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DGDDY
```

For the given values of T, Y, and DY the subroutine should return the partial derivatives, such that

- DGDDY(I,J) contains $\partial g_{\mathrm{I}}(\mathrm{T},\mathrm{Y},\mathrm{DY})/\partial y'_{\mathrm{J}}$ if M is a full matrix (NLM = NEQN);
- DGDDY(I-J+NUM+1,J) contains $\partial g_{\mathrm{I}}(\mathrm{T},\mathrm{Y},\mathrm{DY})/\partial y'_{\mathrm{J}}$ if M is a band matrix ($0 \leq$ NLM $<$ NEQN) (LAPACK / LINPACK / BLAS storage).

You must declare the name MEVAL in an external statement in your program that calls PSIDE.

LDM denotes the leading dimension of M.

All other parameters have the same meaning as within subroutine PSIDE.

T

On entry, T must specify t_0, the initial value of the independent variable.
On successful exit (IDID .EQ. 1), T contains TEND.
On an error return, T is the point reached.

TEND

On entry, TEND must specify the value of the independent variable at which the solution is desired.

RTOL and ATOL

You must assign relative RTOL and absolute ATOL error tolerances to tell the code how small you want the local errors to be. You have two choices

- both RTOL and ATOL are scalars (set IWORK(1) = 0): the code keeps, roughly, the local error of Y(I) below RTOL*ABS(Y(I))+ATOL;
- both RTOL and ATOL are vectors (set IWORK(1) = 1): the code keeps the local error of Y(I) below RTOL(I)*ABS(Y(I))+ATOL(I).

In either case all components must be non-negative.

`IND`

If `IWORK(2) .EQ. 1` , then `IND` should be declared of length `NEQN` and `IND(I)` must specify the index of variable `I`. If `IWORK(2) .EQ. 0` , then `IND` is not referenced and the problem is assumed to be of index 1.
See §12.4 for information how to determine the index of variables of certain problem classes.

`LRWORK`

On entry `LRWORK` must specify the length of the `RWORK` array. You must have for the full partial derivatives case (when `NLJ` = `NEQN`)

```
LRWORK .GE. 20+26*NEQN + 6*NEQN**2,
```

for the case where M is banded and J is full (when `NLJ` = `NEQN` and `NLM` < `NEQN`)

```
LRWORK .GE. 20+(26 + NLM+NUM+1 + 5*NEQN)*NEQN,
```

and for the case where both partial derivatives are banded (when `NLJ` < `NEQN`)

```
LRWORK .GE. 20+(26 + NLJ+NUJ+MLM+NUM+2 + 4*(2*NLJ+NUJ+1))*NEQN.
```

`RWORK`

Real work array of length `LRWORK`. `RWORK(1),...,RWORK(20)` serve as parameters for the code. For standard use, set `RWORK(1),...,RWORK(20)` to zero before calling.

On entry:

- if `RWORK(1) .GT. 0D0` then PSIDE will use `RWORK(1)` as initial stepsize instead of determining it internally.

On exit:

- `RWORK(1)` contains the stepsize used on the last successful step.

`LIWORK`

On entry `LIWORK` must specify the length of the `IWORK` array. You must have

```
LRWORK .GE. 20 + 4*NEQN.
```

`IWORK`

Integer work array of length `LIWORK`. `IWORK(1),...,IWORK(20)` serve as parameters for the code. For standard use, set `IWORK(1),...,IWORK(20)` to zero before calling.

On entry:

- if `IWORK(1) .EQ. 1` then `RTOL` and `ATOL` are vectors instead of scalars,
- if `IWORK(2) .EQ. 1` then `IND` is a vector,

- set IWORK(10) = 0 if PSIDE is called for the first time; for subsequent calls of PSIDE do not reinitialize the parameters IWORK(10),...,IWORK(19) to zero.

On exit:

- IWORK(10) contains the number of successive PSIDE calls,
- IWORK(11) contains the number of g evaluations,
- IWORK(12) contains the number of J and M evaluations (J and M are computed in tandem and count as 1),
- IWORK(13) contains the number of LU-decompositions.
- IWORK(14) contains the number of forward/backward solves,
- IWORK(15) contains the total number of steps (including rejected steps),
- IWORK(16) contains the number of rejected steps due to error control,
- IWORK(17) contains the number of rejected steps due to Newton failure,
- IWORK(18) contains the number of rejected steps due to excessive growth of the solution,
- IWORK(19) contains the number of rejected steps due to IERR .EQ. -1 return of GEVAL.

The integration characteristics in IWORK(11),...,IWORK(14) refer to an implementation on a one-processor computer. When implemented on a parallel computer with four processors, one may divide these numbers by four to obtain the number of sequential evaluations, decompositions and solves.

RPAR and IPAR

RPAR and IPAR are real and integer arrays which you can use for communication between your calling program and the subroutines GEVAL, and/or JEVAL, MEVAL. They are not altered by PSIDE. If you do not need RPAR and IPAR, ignore these parameters by treating them as dummy arguments. If you choose to use them, dimension them in GEVAL and/or JEVAL, MEVAL as arrays of appropriate length. Because of the parallel implementation of PSIDE, GEVAL must not alter RPAR and IPAR to prevent concurrent updating. JEVAL and MEVAL may alter them.

IDID

On exit:

- if IDID .EQ. 1 then the integration was successful,
- if IDID .EQ. -1 then PSIDE could not reach TEND because the stepsize became too small,
- if IDID .EQ. -2 then something else went wrong. For example this happens when the input was invalid. A description message will be printed.

12.4 Index determination

As mentioned before, it is important for higher-index problems to set the index of the variables in the vector `IND`. In this section we specify for certain problem classes, which can easily be written in the form (12.1), how this should be done. The results were taken from [HLR89]. For higher-index problems in these classes we also list the additional conditions that have to be fulfilled by the initial values. We refer to Chapter 11 for information on how PSIDE uses `IND`. If ϕ is a function of q, then we will denote the (partial) derivative of ϕ with respect to q by ϕ_q.

12.4.1 ODEs

First of all, Ordinary Differential Equations (ODEs), which are of the form

$$\begin{aligned} &y' = f(t,y)\,, \quad && y, f \in \mathbb{R}^d, \\ &t_0 \le t \le t_{\text{end}}, && y(t_0) = y_0\,, \end{aligned}$$

are of index 1, i.e. we can set `IWORK(2)` $= 0$.

12.4.2 DAEs of index 1

The class of Differential–Algebraic Equations (DAEs) takes the form

$$\begin{aligned} y' &= f(t,y,z)\,, && y, f \in \mathbb{R}^{d_1}, \\ 0 &= g(t,y,z)\,, && z, g \in \mathbb{R}^{d_2}, \\ t_0 \le t \le t_{\text{end}}, & && y(t_0) = y_0\,, \quad z(t_0) = z_0\,, \end{aligned} \tag{12.2}$$

where y_0 and z_0 are such that $g(t_0, y_0, z_0) = 0$. If g_z is invertible in the neighborhood of the solution, then (12.2) is of index 1 and `IWORK(2)` $= 0$ is the right setting.

12.4.3 IDEs with invertible mass matrix

Also of index 1 are problems of the form

$$\begin{aligned} &M(y)y' = f(t,y)\,, && y, f \in \mathbb{R}^d, \\ &t_0 \le t \le t_{\text{end}}, && y(t_0) = y_0\,, \end{aligned}$$

where $M(y)$ (often called the *mass matrix*) is invertible in the neighborhood of the solution. Again, set `IWORK(2)` $= 0$.

12.4.4 DAEs of index 2

An often arising subclass of (12.2) where g_y is *not* invertible is

$$\begin{aligned} y' &= f(t,y,z)\,, && y, f \in \mathbb{R}^{d_1}, \\ 0 &= g(t,y)\,, && z, g \in \mathbb{R}^{d_2}, \\ t_0 \le t \le t_{\text{end}}, & && y(t_0) = y_0\,, \quad z(t_0) = z_0\,, \end{aligned} \tag{12.3}$$

where y_0 and z_0 are such that $g(t_0, y_0) = 0$ and $g_y(t_0, y_0) f(t_0, y_0, z_0) = 0$. If $g_y f_z$ is invertible in the neighborhood of the solution, then (12.3) is of index 2. The variables y and z are of index 1 and 2, respectively, so set `IND(I)` $= 1$ if `I` corresponds to a y-component, and `IND(I)` $= 2$ if `I` corresponds to a z-component.

12.4.5 IDEs of index 3

If the problem is of the form

$$\begin{array}{lll} y' = f(t,y,z), & y,f \in \mathbb{R}^{d_1}, & \\ z' = k(t,y,z,u), & z,k \in \mathbb{R}^{d_2}, & \\ 0 = g(t,y), & u,g \in \mathbb{R}^{d_3}, & \\ t_0 \le t \le t_{\text{end}}, & y(t_0) = y_0\,, \quad z(t_0) = z_0\,, & u(t_0) = u_0\,, \end{array} \tag{12.4}$$

where $g_y f_z k_u$ is invertible in the neighborhood of the solution and y_0, z_0 and u_0 satisfy the conditions

$$\begin{aligned} & g(t_0,y_0) = 0\,, \\ & g_y(t_0,y_0)f(t_0,y_0,z_0) = 0\,, \\ & g_{yy}(t_0,y_0)(f(t_0,y_0,z_0),f(t_0,y_0,z_0))+ \\ & \quad g_y(t_0,y_0)(f_y(t_0,y_0,z_0)f(t_0,y_0,z_0) + f_z(t_0,y_0,z_0)k(t_0,y_0,z_0)) = 0\,, \end{aligned}$$

then (12.4) is an IDE of index 3. The variables y, z and u are of index 1, 2 and 3, respectively, so set `IND(I)` $= 1$ if `I` corresponds to a y-component, `IND(I)` $= 2$ if `I` corresponds to a z-component, and `IND(I)` $= 3$ if `I` corresponds to a u-component.

12.4.6 Multibody systems of index 3

In mechanics one often encounters the problem

$$\begin{array}{lll} q' = u\,, & q,u \in \mathbb{R}^{d_1}, & \\ M(q)u' = f(t,q,u) + G^{\mathrm{T}}(q)\lambda\,, & f \in \mathbb{R}^{d_2}, & \\ 0 = g(t,q)\,, & \lambda,g \in \mathbb{R}^{d_3}, & \\ t_0 \le t \le t_{\text{end}}, & q(t_0) = q_0\,, \quad u(t_0) = u_0\,, & \lambda(t_0) = \lambda_0\,, \end{array} \tag{12.5}$$

where $G(q) = g_q$, the matrix $M(q)$ non-singular in the neighborhood of the solution and q_0, u_0 and λ_0 are such that they satisfy

$$\begin{aligned} & g(t_0,q_0) = 0\,, \\ & G(t_0,q_0)u_0 = 0\,, \\ & g_{qq}(t_0,q_0)(u_0,u_0) + G(t_0,q_0)M^{-1}(q_0)(f(t_0,q_0,u_0) + G^{\mathrm{T}}(q_0)\lambda_0) = 0\,. \end{aligned}$$

We could rewrite the system to the form (12.4) by premultiplying both sides of the u'-equation by $M^{-1}(q)$. Consequently, (12.5) is of index 3 and the variables q, u and λ are of index 1, 2 and 3, respectively, so set `IND(I)` $= 1$ if `I` corresponds to a q-component, `IND(I)` $= 2$ if `I` corresponds to a u-component, and `IND(I)` $= 3$ if `I` corresponds to a λ-component.

12.5 Example

Here we give a simple example, solving the Van der Pol problem, an ODE of dimension 2.

12.5.1 Driver for Van der Pol problem

```
      program psidex
c
c PSIDE example: Van der Pol problem
c
c  - ODE of dimension 2                                    y' = f
c  - formulated as general IDE                            g = f - y' = 0
c  - analytical partial derivative J (full 2x2 matrix)    dg/dy = df/dy
c  - analytical partial derivative M (band matrix)        dg/dy' = -I
c
      integer neqn,nlj,nuj,nlm,num
      logical jnum,mnum
      parameter (neqn=2,nlj=neqn,nuj=neqn,nlm=0,num=0)
      parameter (jnum=.false., mnum=.false.)
      integer lrwork, liwork
      parameter (lrwork = 20+26*neqn+6*neqn**2, liwork = 20+4*neqn)

      integer ind,iwork(liwork),ipar,idid
      double precision y(neqn),dy(neqn),t,tend,rtol,atol,
     +                 rwork(lrwork),rpar

      external vdpolg,vdpolj,vdpolm

      integer i

c initialize PSIDE

      do 10 i=1,20
         iwork(i) = 0
         rwork(i) = 0d0
   10 continue

c consistent initial values

      t      =  0d0
      y(1)   =  2d0
      y(2)   =  0d0
      dy(1)  =  0d0
      dy(2)  = -2d0

      tend   = 41.5d0
```

```
c set scalar tolerances

      rtol = 1d-4
      atol = 1d-4

      write(*,'(1x,a,/)') 'PSIDE example solving Van der Pol problem'

      call pside(neqn,y,dy,vdpolg,
     +           jnum,nlj,nuj,vdpolj,
     +           mnum,nlm,num,vdpolm,
     +           t,tend,rtol,atol,ind,
     +           lrwork,rwork,liwork,iwork,
     +           rpar,ipar,idid)

      if (idid.eq.1) then
         write(*,'(1x,a,f5.1,/)')  'solution at t = ', tend
         write(*,'(1x,a,e11.3)')   '   y(1) =', y(1)
         write(*,'(1x,a,e11.3,/)') '   y(2) =', y(2)
         write(*,'(1x,a,i4)')      'number of steps =', iwork(15)
         write(*,'(1x,a,i4)')      'number of f-s   =', iwork(11)
         write(*,'(1x,a,i4)')      'number of J-s   =', iwork(12)
         write(*,'(1x,a,i4)')      'number of LU-s  =', iwork(13)
      else
         write(*,'(1x,a,i4)') 'PSIDE failed: IDID =', idid
      endif

      end

      subroutine vdpolg(neqn,t,y,dy,g,ierr,rpar,ipar)
      integer neqn,ierr,ipar(*)
      double precision t,y(neqn),dy(neqn),g(neqn),rpar(*)
      g(1) = y(2)-dy(1)
      g(2) = 500d0*(1d0-y(1)*y(1))*y(2)-y(1)-dy(2)
      return
      end

      subroutine vdpolj(ldj,neqn,nlj,nuj,t,y,dy,dgdy,ierr,rpar,ipar)
      integer ldj,neqn,nlj,nuj,ierr,ipar(*)
      double precision t,y(neqn),dy(neqn),dgdy(ldj,neqn),rpar(*)
      dgdy(1,1) = 0d0
      dgdy(1,2) = 1d0
      dgdy(2,1) = -1000d0*y(1)*y(2)-1d0
```

```
      dgdy(2,2) = 500d0*(1d0-y(1)*y(1))
      return
      end

      subroutine vdpolm(ldm,neqn,nlm,num,t,y,dy,dgddy,ierr,rpar,ipar)
      integer ldm,neqn,nlm,num,ierr,ipar(*)
      double precision t,y(neqn),dy(neqn),dgddy(ldm,neqn),rpar(*)
      dgddy(1,1) = -1d0
      dgddy(1,2) = -1d0
      return
      end
```

12.5.2 Output for Van der Pol problem

This is the output of the example given in the previous subsection.

```
PSIDE example solving Van der Pol problem

solution at t =  41.5

   y(1) =  0.194E+01
   y(2) = -0.140E-02

number of steps =  22
number of f-s   = 218
number of J-s   =   9
number of LU-s  =  88
```

12.6 PSIDE and the 'Test set for IVP solvers'

A user-friendly interface for PSIDE is supplied by the 'Test set for IVP solvers', which is available via the World Wide Web [LSV96]. This test platform contains not only the Fortran 77 routines for many test problems in a uniform format, but also drivers that link this format to the solvers RADAU5 [HW96a], VODE [BHB92], DASSL [Pet91] and PSIDE. This means that if one wants to solve a particular set of differential equations with PSIDE, it suffices to write the code that defines the problem in the test set format and link it to the solver and the test set driver. Without using the test set, one would have to write a driver oneself and bother about the right input parameters, the dimension of the work arrays, the output format, etc.

Another advantage is that it is easy to compare solvers mutually. To give an impression of the performance of PSIDE in relation to that of the other solvers, we give in

Figure 12.1–12.2 work-precision diagrams for two problems from [LSV96]. They correspond to the Medical Akzo Nobel problem, a set of semi-discretized partial differential equations of dimension 400 which describe the injection of a medicine in a tumorous tissue, and the NAND gate, a set of 14 implicit differential equations of index 1 which model a electrical circuit performing the logical `NOT(AND)` operation. To produce these diagrams, we used for every solver a range of input tolerances, measured the accuracy delivered by the solver in number of correct digits and plotted these numbers against the CPU times needed for the runs on a logarithmic scale. The PSIDE-1 curves correspond to timings on a one-processor machine, the PSIDE-4 curves were obtained by dividing the one-processor timings by the speed-up factors on four processors. For an explanation how these factors were obtained, we refer to [LSV96] or §6.5. Results of VODE and RADAU5 are not included in Figure 12.2, because these solvers can not handle implicit differential equations directly. From Figure 12.1 we see that for the Medical Akzo Nobel problem PSIDE is on one processor about as efficient as VODE and DASSL and less efficient than RADAU5, whereas PSIDE using four processors is the most efficient solver. Figure 12.2 reveals that for the NAND gate DASSL performs better than PSIDE in one-processor mode, but worse if four processors are used. These figures are quite representative for the numerous comparisons in [LSV96], which show that the speed-up factor of PSIDE with respect tot the other solvers is between 1.5 and 3.8, depending on problem and solver.

Integration characteristics, complete descriptions of the test problems and the test set format, full details about the work-precision diagrams, as well as comparisons for other test problems, can be found in [LSV96].

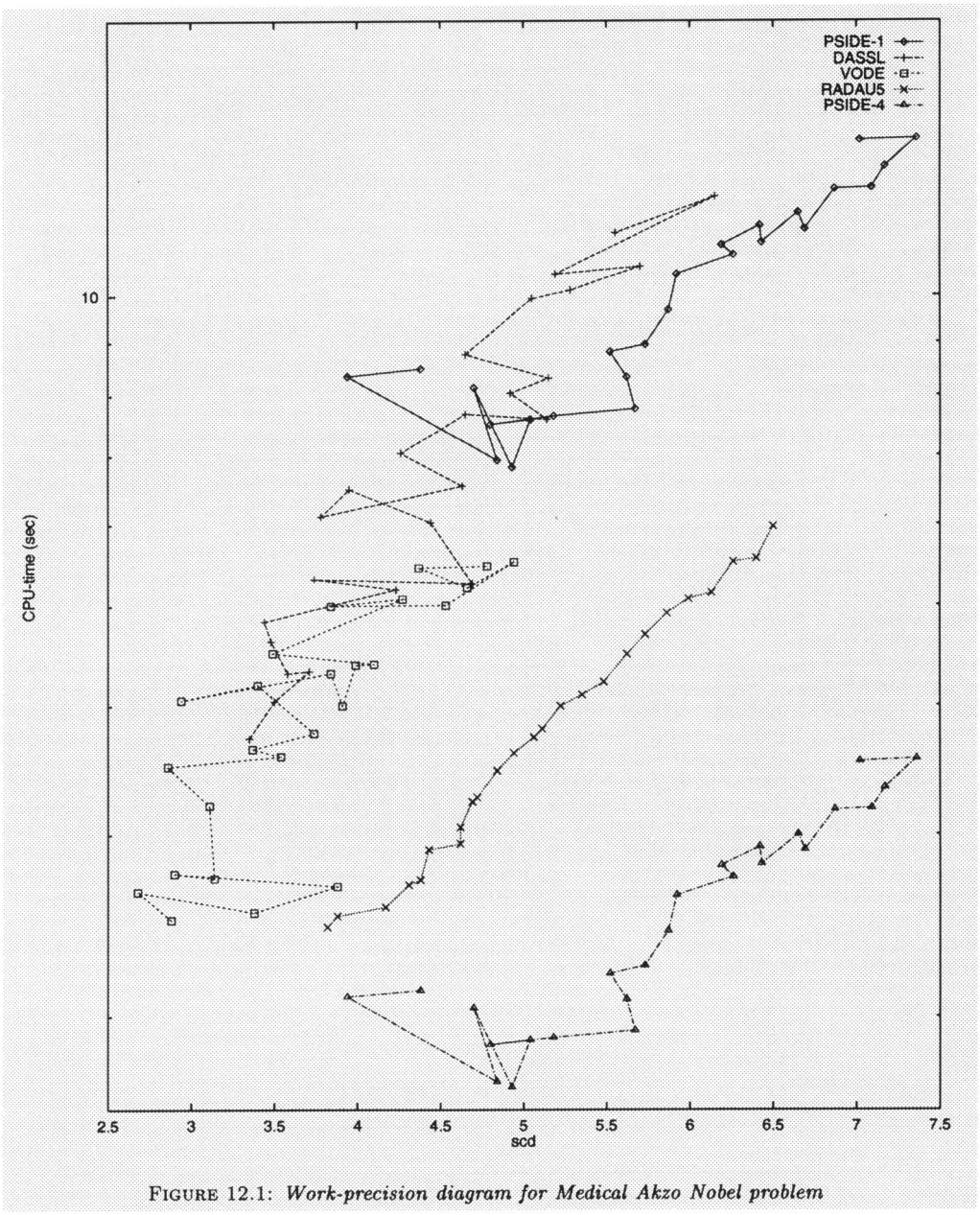

FIGURE 12.1: *Work-precision diagram for Medical Akzo Nobel problem*

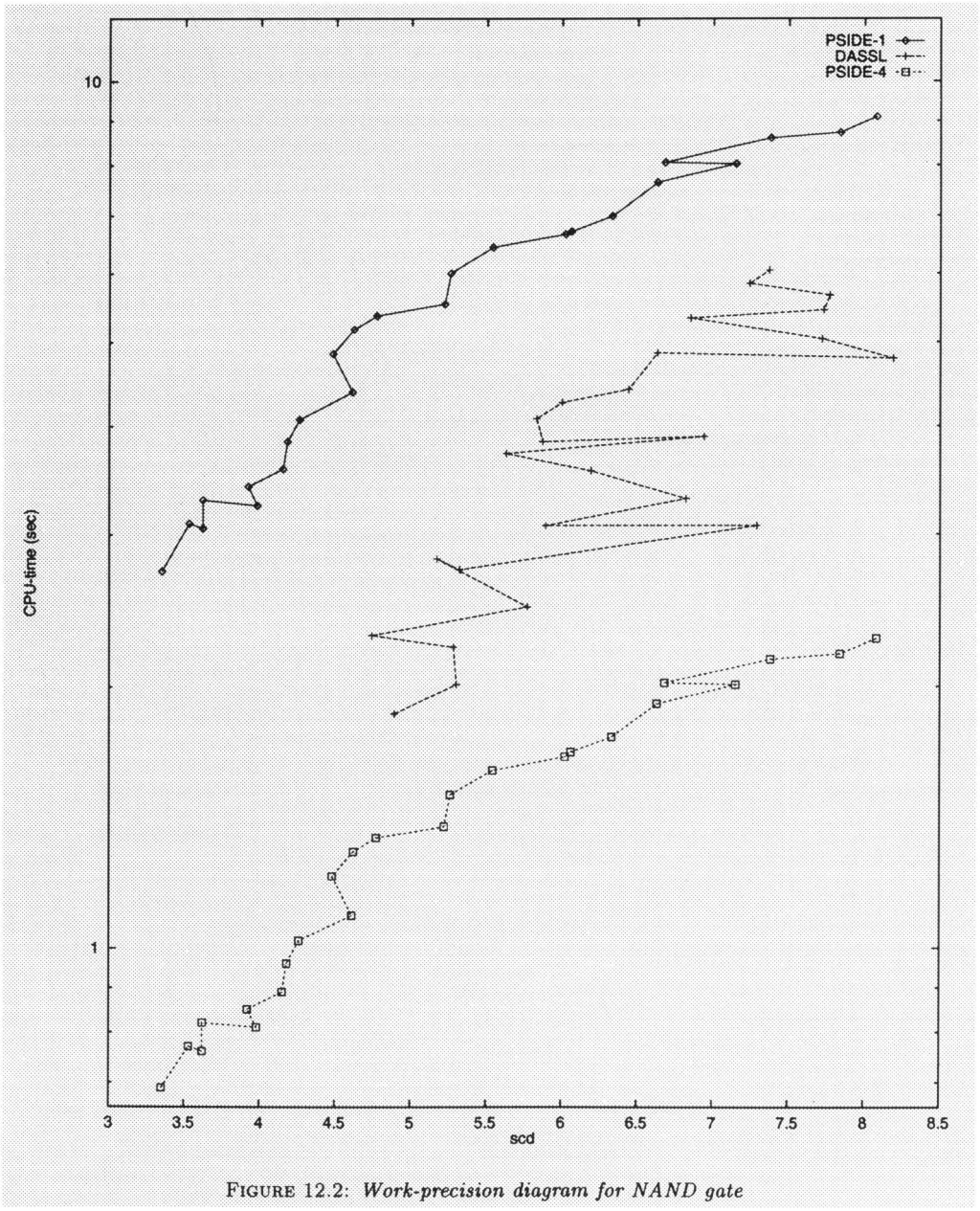

FIGURE 12.2: *Work-precision diagram for NAND gate*

Acknowledgements

First of all I want to express my deep feelings of gratitude towards my promotor Piet van der Houwen. He stimulated me enormously to work hard, enthusiastically and productively, by holding himself up as an example and by always sharing the flood of his never-running-dry source of ideas with me.

I am also very grateful to Ben Sommeijer for introducing me so patiently and gently to the subject and for always being available for discussions, questions and proofreading, despite his many activities. His constructive criticism improved the quality of this thesis considerably.

It was a great pleasure for me to have collaborated with so many different people while writing the papers collected in this thesis. Therefore I want to acknowledge explicitly Joke Blom, Walter Hoffmann, Piet van der Houwen, Walter Lioen, Eleonora Messina, János Pintér, Gustaf Söderlind, Ben Sommeijer, Walter Stortelder and Wolter van der Veen. They are the co-authors of this thesis and all of them taught me different aspects of doing research and reporting on it.

Of course this thesis could not be realized without the support of *Technologiestichting STW*. This foundation sponsored the project 'Parallel codes for circuit analysis and control engineering', of which this thesis is one of the deliverables. I thank STW for enabling me to conduct the research and to attend all the main conferences in the area. The mission statement of this foundation, not only doing scientific research, but also focusing on industrial applications, is attractive to me, and I hope to continue in this direction in the future.

I also feel indebted to the *Centrum voor Wiskunde en Informatica,* for offering me a working environment with excellent facilities and many nice colleagues.

Although the word 'synergy' is often abused, this is not the case in relation to the group, working on the aforementioned project, consisting of Piet, Ben, Walter Lioen, Walter Hoffmann, Wolter, myself and, for a shorter period, Eleonora. From this group of—in some respects—complementary characters evolved a continuous stimulus to proceed and reach new goals. Thus it was hardly possible to 'get stuck' in the research, a phenomenon that is to my opinion one of the greatest hazards in doing a PhD. I will surely miss the lively discussions in the weekly meetings of our project group.

Walter Hoffmann contributed considerably as co-promotor to this thesis as well. Since the numerical treatment of differential equations is not his main research topic, his comments prevented us from becoming narrow-minded. Furthermore, his perfectionism in the use of language and notation is—hopefully—reflected in the underlying work.

Very fruitful and pleasant was the three week period that I spent in Lund on invitation of Gustaf Söderlind. Here I learned a lot about the software aspects of numerically solving differential equations. I thank Gustaf for his friendly cooperation and sincerely hope that

we will continue to work together in the future.

It is difficult not to underestimate the importance of the people who help you when you encounter any computer problems. Computer advice is a very unbalanced form of human interaction. The questioner benefits from it, whereas the consultant is only disturbed in his own activities. Walter and Joke, thank you very much for transforming me from a computer illiterate into person with a computer knowledge about a quarter as large as yours.

The ED&T AS group at Philips Research Laboratories was so kind to enable me to work on their circuit simulator during a traineeship in the summer of 1997. This single phrase does not express sufficiently my gratitude towards Theo and Anneke van Dijk, who offered me so much hospitality in Waalre during this traineeship. Theo served as advisor on lay-out aspects as well.

I thank Tobias Baanders not only for the design of the cover of this thesis, but also for being a nice (skating) colleague and for his help and interest in making material for presentations of the research at workshops and conferences (and even T-shirts). For producing Figure 10.1 on page 147, I am grateful to Robert van Liere. It is undoubtedly the nicest figure in the thesis. Roel Verlaan, Jeroen van Gaart, Hélène de Swart, Priscella Kastawi and Jason Frank were so kind to proofread parts of the manuscript carefully.

Finally, I want to thank my family and friends for at least pretending to be interested in my research.

Amsterdam, October 1997

Samenvatting (Summary in Dutch)

Stelsels impliciete differentiaalvergelijkingen komt men tegen bij het modelleren van vele tijdsafhankelijke industriële processen, zoals het gedrag van electrische circuits, de bewegingen van robots en chemische reacties. Omdat de stelsels vrijwel altijd te moeilijk zijn om exact op te lossen, gebruikt men meestal een numeriek schema om met behulp van een computer een benadering voor de oplossing te vinden.

Vandaag de dag neemt de omvang en complexiteit van de problemen sneller toe dan de snelheid van de rekenprocessoren. Om de rekentijd toch terug te brengen biedt de komst van parallelle computers, die meerdere processoren bevatten, een uitkomst. Dit proefschrift gaat over het ontwerp van numerieke schema's voor dit type computer, waarbij de verdeling van het werk over de verschillende processoren onafhankelijk is van het probleem. Dit is geen eenvoudige taak omdat het oplosproces zeer sequentieel van aard is; veel conventionele methoden hebben om de benadering van de oplossing in een tijdspunt te kunnen berekenen alle informatie van het vorige tijdspunt nodig.

Een interessante klasse van methoden met potentie om toch uiteen te vallen in deeltaken die tegelijkertijd kunnen worden uitgevoerd, is die van de Runge–Kutta-methoden. Deze methoden berekenen meerdere benaderingen per tijdspunt, die men ook wel stagewaarden noemt. In dit proefschrift hebben we ons gericht op het dusdanig ontwerpen en/of aanpassen van Runge–Kutta-methoden dat de stagewaarden onafhankelijk van elkaar berekend kunnen worden. Daarbij is onderscheid gemaakt op grond van de stijfheid van een probleem.

Stijve problemen worden gekarakteriseerd door sterk variërende tijdschalen van de verschillende oplossingscomponenten. Voor niet-stijve problemen hebben we een code ontwikkeld gebaseerd op een gegeneraliseerde Runge–Kutta-methode, toegepast met een vastpunts iteratieproces. Deze code is op vijf processoren tot ruim drie keer sneller dan de code DOPRI8, die over het algemeen beschouwd wordt als de beste sequentiële solver voor niet-stijve problemen.

Voor stijve problemen geeft het bovengenoemde schema aanleiding tot numerieke instabiliteit. Dat wil zeggen dat de fout in de numerieke benadering al snel overvleugeld wordt door een opeenstapeling van allerlei fouten. De familie van impliciete Runge–Kutta-methoden kent dit probleem niet, maar deze methoden zijn veel duurder omdat ze de oplossing vereisen van niet-lineaire stelsels, waarvan de dimensie gelijk is aan het aantal stagewaarden maal de probleemdimensie. We hebben twee technieken ontworpen om deze stelsels goedkoop op te lossen met behulp van parallelle computers.

In de eerste aanpak lossen we de niet-lineaire vergelijkingen op met een iteratief proces dat zo gemaakt is dat een stagewaarde onafhankelijk van de andere opgelost kan worden uit lineaire stelsels, waarvan de dimensie gelijk is aan de probleemdimensie. Door nu voor elke stagewaarde één processor beschikbaar te stellen hebben we een parallel

proces verkregen. De basis voor deze techniek is de benadering van de matrix die de Runge–Kutta-methode vastlegt door een benedendriehoeksmatrix met positieve diagonaalelementen. Voor een groot aantal Runge–Kutta-methoden hebben we bewezen dat zo'n driehoeksmatrix eenvoudig te construeren is.

De tweede aanpak gebruikt een gemodificeerd Newton proces om de grote stelsels niet-lineaire vergelijkingen terug te brengen tot lineaire stelsels van dezelfde dimensie. Vervolgens benaderen we de oplossing van deze lineaire stelsels met behulp van een nieuw iteratieproces dat voor elke stagewaarde de oplossing van lineaire stelsels van de probleemdimensie vereist. Wederom is het mogelijk om deze stelsels tegelijkertijd op te lossen. We hebben verschillende varianten uitgewerkt, alsmede enkele generalisaties geconstrueerd naar de klasse van meerstaps Runge–Kutta-methoden. Hoewel deze techniek ingewikkelder is dan de eerste, werkt hij uiteindelijk efficiënter. Bovendien kunnen we, afhankelijk van de variant, sterke resultaten afleiden over de convergentie van het nieuwe iteratieproces.

Beide technieken leiden tot een groot aantal lineaire stelsels van de probleemdimensie en de efficiëntie van het uiteindelijke proces hangt in belangrijke mate af van hoe deze opgelost worden. De matrices van de stelsels bezitten een speciale structuur met vaak veel nullen. Met een oplosmethode die speciaal is ontwikkeld voor deze vorm is het gelukt om de stelsels aanzienlijk sneller op te lossen.

Gebaseerd op de tweede aanpak hebben we de code PSIDE (de afkorting van de titel van dit proefschrift) ontwikkeld. PSIDE is een robuuste code van zevende orde en geschikt voor computers met vier processoren. De klasse van problemen waarvoor PSIDE gebruikt kan worden is ruimer dan die voor de meeste bestaande codes. Voor sommige problemen speelt het begrip index een rol. Een probleem dat van een hogere index is, heeft oplossingscomponenten die extra gevoelig zijn voor verstoringen. PSIDE kan enkele veel voorkomende klassen van hogere index problemen oplossen. Tevens is bijzondere aandacht besteed aan het ontwerpen van een foutschatter die ook voor zeer stijve problemen robuust werkt.

We hebben PSIDE getest op een uitgebreide verzameling problemen en vergeleken met enkele veelgebruikte codes. Op één processor is PSIDE meestal langzamer dan de andere codes en soms ongeveer even snel. Voor de implementatie van PSIDE op vier processoren worden speed-up factoren gehaald ten opzichte van de andere codes variërend van 1,5 tot 3,8, afhankelijk van probleem en code.

Verder komt een toepassing uit de globale optimalisatie aan bod. Het gaat om het Fekete probleem, waarin een aantal punten dusdanig over een bol verspreid moet worden dat hun onderlinge afstanden zo groot mogelijk zijn. Het blijkt dat dit probleem geschreven kan worden als een stelsel impliciete differentiaalvergelijkingen. Omdat de complexiteit van het probleem snel groeit bij een toenemend aantal punten, is dit bij uitstek een geschikte toepassing voor PSIDE, dat aanzienlijk betere resultaten laat zien dan een globale optimalisator.

References

[BAR87] L.G. Birta and O. Abou-Rabia. Parallel block predictor-corrector methods for ODEs. *IEEE Trans. Comput.*, C-36:299–311, 1987.

[BC89] K. Burrage and F.H. Chipman. Construction of A-stable diagonally implicit multiwave methods. *SIAM J. Numer. Anal.*, 26:391–413, 1989.

[BCP89] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations.* North–Holland, New York–Amsterdam–London, 1989.

[Bel87] A. Bellen. *Parallelism across the steps for difference and differential equations*, pages 22–35. Lecture Notes in Mathematics 1386. Springer-Verlag, 1987.

[Ben96] C. Bendtsen. *Parallel Numerical Algorithms for the Solution of Systems of Ordinary Differential Equations.* PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, June 1996.

[BHB92] Peter N. Brown, Alan C. Hindmarsh, and George D. Byrne. *VODE: A variable coefficient ODE solver*, August 1992. Available at `http://www.netlib.org/ode/vode.f`.

[Bin85] Zhou Bing. A-stable and L-stable block implicit one-block methods. *Journal of Computational Mathematics*, 3(4):328–341, 1985.

[BJZ90] A. Bellen, Z. Jackiewicz, and M. Zennaro. Time-point relaxation Runge–Kutta methods for ordinary differential equations. *J. Comp. Appl. Math.*, 45:121–138, 1990.

[BS97] K. Burrage and H. Suhartanto. Parallel iterated methods based on multistep Runge–Kutta methods of Radau type. *Advances in Computational Mathematics*, 7:37–57, 1997.

[Bur78] K. Burrage. A special family of Runge–Kutta methods for solving stiff differential equations. *BIT*, 18:373–383, 1978.

[Bur93a] K. Burrage. Parallel methods for initial value problems. *Applied Numerical Mathematics*, 11:5–26, 1993.

[Bur93b] K. Burrage. The search for the Holy Grail, or: Predictor-corrector methods for solving ODEIVPs. *Appl. Numer. Math.*, 11:125–141, 1993.

[Bur95] K. Burrage. *Parallel and Sequential Methods for Ordinary Differential Equations.* Clarendon Press, Oxford, 1995.

[But76] J.C. Butcher. On the implementation of implicit Runge–Kutta methods. *BIT*, 16:237–240, 1976.

[But87] J.C. Butcher. *The numerical analysis of ordinary differential equations, Runge–Kutta and general linear methods.* Wiley, New York, 1987.

[BVZ90] A. Bellen, R. Vermiglio, and M. Zennaro. Parallel ODE-solvers with stepsize control. *JCAM*, 31:277–293, 1990.

[CB83] G.J. Cooper and J.C. Butcher. An iteration scheme for implicit Runge–Kutta methods. *IMA J. Numer. Anal.*, 3:127–140, 1983.

[CGG+91] 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.

[CH87] M.T. Chu and H. Hamilton. Parallel solution of ODEs by multi-block methods. *SIAM J. Statist. Comput.*, 8:342–353, 1987.

[Cha93] P. Chartier. *Parallelism in the numerical solutions of initial value problems for ODEs and DAEs.* PhD thesis, Université de Rennes I, France, 1993.

[Cra94a] Cray Research, Inc. *CF77 Commands and Directives*, SR-3771 6.0 edition, 1994.

[Cra94b] Cray Research, Inc. *UNICOS Performance Utilities Reference Manual*, SR-2040 8.0 edition, 1994.

[DER86] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Monographs on Numerical Analysis. Oxford Science Publications, 1986.

[DG87] J.J. Dongarra and E. Grosse. Distribution of software via electronic mail. Technical Report 30, 403–407, Commun. ACM, 1987. (netlib@research.att.com).

[Duf77] I.S. Duff. MA28 - a set of Fortran subroutines for sparse unsymmetric linear equations. Technical Report AERE R8730, HMSO, London, 1977.

[EHL75] W.H. Enright, T.E. Hull, and B. Lindberg. Comparing numerical methods for stiff systems of ODEs. *BIT*, 15:10–48, 1975.

[Fek23] M. Fekete. Über die Verteilung der Wurzeln bei gewisser algebraischen Gleichungen mit ganzzahligen Koeffizienten. *Mathematische Zeitschrift*, 17, 1923.

[Gea69] C.W. Gear. The automatic integration of stiff ordinary differential equations. In *Proceedings of the IFIP Congress 1968*, pages 187–193, Amsterdam, 1969. North Holland.

[GL89] G.H. Golub and C.F. van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore and London, second edition, 1989.

[Got77] B.A. Gottwald. MISS - ein einfaches Simulations-System für biologische und chemische Prozesse. *EDV in Medizin und Biologie*, 3:85–90, 1977.

[GS69] 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.

[GS97] K. Gustafsson and G. Söderlind. Control strategies for the iterative solution of nonlinear equations in ODE solvers. *SIAM Journal on Scientific and Statistical Computing*, 18(1):23–40, Jan 1997.

[Gus92] K. Gustafsson. *Control of Error and Convergence in ODE Solvers.* PhD thesis, Lund Institute of Technology, 1992.

[GX93] C.W. Gear and Xu Xuhai. Parallelism across time in ODEs. *Appl. Numer. Math.*, 11:45–68, 1993.

[Hin83] Alan C. Hindmarsh. ODEPACK, a systemized collection of ODE solvers. In R. Stepleman et al., editors, *Scientific Computing*, pages 55–64, Amsterdam, 1983. IMACS, North-Holland Publishing Company.

[HLR89] 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.

[HM96] 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.

[HNW87] E. HAIRER, S.P. NØRSETT, AND G. WANNER. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, 1987.

[HNW93] E. HAIRER, S.P. NØRSETT, AND G. WANNER. *Solving Ordinary Differential Equations I: Nonstiff Problems.* Springer-Verlag, second revised edition, 1993.

[Hor76] 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.

[HS90] P.J. VAN DER HOUWEN AND B.P. SOMMEIJER. Parallel iteration of high-order Runge–Kutta methods with stepsize control. *JCAM*, 29:111–127, 1990.

[HS91] 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.

[HS92] P.J. VAN DER HOUWEN AND B.P. SOMMEIJER. Block Runge–Kutta methods on parallel computers. *ZAMM*, 72:3–18, 1992.

[HS93] P.J. VAN DER HOUWEN AND B.P. SOMMEIJER. Analysis of parallel diagonally implicit iteration of Runge–Kutta methods. *Applied Numerical Mathematics*, 11:169–188, 1993.

[HS96] P.J. VAN DER HOUWEN AND B.P. SOMMEIJER. CWI contributions to Runge–Kutta methods on parallel computers. *Applied Numerical Mathematics*, 22:327–344, 1996.

[HSV94] 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.

[HSV95] 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.

[HV97] P.J. VAN DER HOUWEN AND W.A. VAN DER VEEN. The solution of implicit differential equations on parallel computers, 1997. Submitted for publication.

[HW91] E. HAIRER AND G. WANNER. *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems.* Springer-Verlag, 1991.

[HW96a] E. HAIRER AND G. WANNER. *RADAU5*, July 1996. Available at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau5.f`.

[HW96b] E. HAIRER AND G. WANNER. *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems.* Springer-Verlag, second revised edition, 1996.

[IN91] A. ISERLES AND S.P. NØRSETT. *Order Stars.* Chapman & Hall, 1991.

[JN95] 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.

[JT94] M.Z. JACOBSON AND R.P. TURCO. SMVGEAR: A sparse-matrix, vectorized Gear code for atmospheric models. *Atmospheric Environment*, 28:273–284, 1994.

[Lie87] I. LIE. Some aspects of parallel Runge–Kutta methods. Technical Report 3/87, Dept. of Mathematics, University of Trondheim, 1987.

[Lio96] W.M. LIOEN. On the diagonal approximation of full matrices. *Journal of Computational and Applied Mathematics*, 75:35–42, 1996.

[LN89] I. LIE AND S.P. NØRSETT. Superconvergence for multistep collocation. *Math. Comput.*, 52:65–79, 1989.

[LPS86] A. LUBOTZKY, R. PHILLIPS, AND P. SARNAK. Hecke operators and distributing points on the sphere I. *Communications in Pure and Applied Mathematics*, 89:149–186, 1986.

[LSV96] 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. Available at `http://www.cwi.nl/cwi/projects/IVPtestset/`.

[Mar57] H.M. MARKOWITZ. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

[ML67] W.L. MIRANKER AND W. LINIGER. Parallel methods for the numerical integration of ordinary differential equations. *Math. Comp.*, 21:303–320, 1967.

[Nev85] 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.

[Nør76] S.P. NØRSETT. Runge–Kutta methods with a multiple real eigenvalue only. *BIT*, 16:388–393, 1976.

[NS89] S.P. NØRSETT AND H.H. SIMONSEN. Aspects of parallel Runge–Kutta methods. In A. Bellen, C.W. Gear, and E. Russo, editors, *Numerical Methods for Ordinary Differential Equations*. Lecture Notes in Mathematics 1386, Springer-Berlin, 1989. Proceedings L'Aquila 1987.

[Ore93] B. OREL. Parallel Runge–Kutta methods with real eigenvalues. *Applied Numerical Mathematics*, 11:241–250, 1993.

[OS96] H. OLSSON AND G. SÖDERLIND. Stage value predictors and efficient Newton iterations in implicit Runge–Kutta methods. 1996. *Submitted for publication.*

[Par95] P.M. PARDALOS. An open global optimization problem on the unit sphere. *Journal of Global Optimization*, 6:213, 1995.

[PD81] P.J. PRINCE AND J.R. DORMAND. High order embedded Runge–Kutta formulae. *J. Comput. Appl. Math.*, 7:67–75, 1981.

[Pet91] L.R. PETZOLD. *DASSL: A Differential/Algebraic System Solver*, June 1991. Available at `http://www.netlib.org/ode/ddassl.f`.

[Pin96] J.D. PINTÉR. *Global Optimization in Action, Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications.* Kluwer Academic Publishers, Dordrecht–Boston–London, 1996.

[Pin97] J.D. PINTÉR. LGO – a program system for continuous and Lipschitz Global Optimization. In I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors, *Developments in Global Optimization*, Dordrecht–Boston–London, 1997. Kluwer Academic Publishers.

[RT92] L. REICHEL AND L.N. TREFETHEN. Eigenvalues and pseudo-eigenvalues of Toeplitz matrices. *Linear Algebra Appl.*, 162/164:153–185, 1992.

[SB84] L.F. SHAMPINE AND L. BACA. Error estimators for stiff differential equations. *Journal of Computational and Applied Mathematics*, 11:197–207, 1984.

[Sch94] 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.

[Som93] B.P. SOMMEIJER. *Parallelism in the numerical integration of initial value problems.* CWI Tract 99, CWI, Amsterdam, 1993.

[SS93] M. SHUB AND S. SMALE. Complexity of Bezout's theorem III. Condition number and packing. *Journal of Complexity*, 9:4–14, 1993.

[SSV97] B.P. SOMMEIJER, L.F. SHAMPINE, AND J.G. VERWER. *RKC*, 1997. Available at `http://www.netlib.org/ode/rkc.f`.

[Tsu59] M. TSUJI. *Potential Theory in Modern Function Theory.* Maruzen, Tokyo, 1959.

[Var62] R.S. VARGA. *Matrix iterative analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1962.

[VBLS95] J.G. VERWER, J.G. BLOM, M. VAN LOON, AND E.J. SPEE. A comparison of stiff ODE solvers for atmospheric chemistry problems. *Atmospheric Environment*, 29, 1995.

[Vee97] W.A. VAN DER VEEN. *Parallelism in the numerical solution of ordinary and implicit differential equations.* PhD thesis, University of Amsterdam, 1997.

[Ver94] J.G. VERWER. Gauss-Seidel iteration for stiff ODEs from chemical kinetics. *SIAM J. Sci. Comput.*, 15(5):1243–1259, 1994.

[ZWS81] ZAHARI ZLATEV, JERZY WASNIEWSKI, AND KJELD SCHAUMBURG. *Y12M, Solution of Large and Sparse Systems of Linear Algebraic Equations.* Lecture Notes in Computer Science No. 121. Springer-Verlag, Berlin Heidelberg New York, 1981.

ISBN: 90-74795-77-3
Cover: Tobias Baanders
Printing & binding: Ponsen en Looijen B.V., Wageningen
No. of copies: 350
Font: Computer Modern Roman
Typesetting: LaTeX 2_ε; style files available on request (jacques@cwi.nl)

Stellingen

behorende bij het proefschrift

"Parallelle Software voor Impliciete Differentiaalvergelijkingen"

door Jacques J.B. de Swart

1. Stel dat het stelsel gewone differentiaalvergelijkingen $y' = f(y(t))$, $y(t_0) = y_0$, met $y, f \in \mathbb{R}^d$, numeriek wordt opgelost met een Runge–Kutta methode van de vorm

$$Y_n = (\mathbb{1} e_s^{\mathrm{T}} \otimes I) Y_{n-1} + h(A \otimes I) F(Y_n), \qquad n = 1, 2, \ldots, \tag{1}$$

waarin Y_n een vector voorstelt met d-dimensionale benaderingen $Y_{n,i}$ voor de oplossing in $t_{n-1} + c_i h$, $i = 1, \ldots, s$, en h de stapgrootte. Van de vector c nemen we aan dat $c_s = 1$. Verder staat $\otimes$ voor het Kronecker product, $\mathbb{1}$ voor de s-dimensionale vector $(1, \ldots, 1)^{\mathrm{T}}$, de vector e_s, die van dezelfde dimensie is, voor $(0, \ldots, 0, 1)^{\mathrm{T}}$, de matrix I voor de d-dimensionale eenheidsmatrix, A voor de $s \times s$ matrix met Runge–Kutta parameters en $F(Y_n)$ voor de vector $(f(Y_{n,1})^{\mathrm{T}}, \ldots, f(Y_{n,s})^{\mathrm{T}})^{\mathrm{T}}$, en is Y_0 gelijk aan $(\mathbb{1} \otimes I) y_0$.

Beschouw het volgende iteratieproces om Y_n uit (1) op te lossen:

$$\begin{array}{rcll} Y_n^{(1)} & \text{gedefinieerd door één of andere predictor,} & \\ Y_n^{(j)} - h(D \otimes I) F(Y_n^{(j)}) & = & \mathbb{1} e_s^{\mathrm{T}} \otimes Y_{n-1}^{(j+j^*-1)} + h((A-D) \otimes I) F(Y_n^{(j-1)}), & j = 2, \ldots, m, \\ Y_n^{(j)} & = & Y_n^{(m)}, & j > m, \end{array}$$

met j^* een geheel getal groter dan nul.

Voor dit proces zijn de volgende twee uitspraken waar:

(i) De iteranden $Y_n^{(j)}, Y_{n-1}^{(j+j^*)}, Y_{n-2}^{(j+2j^*)}, \ldots, Y_1^{(j+nj^*-j^*)}$ kunnen tegelijkertijd worden uitgerekend.

(ii) Voor de lineaire testvergelijking $y' = \lambda y$ convergeert het proces naar de echte oplossing van (1) als $\rho(h\lambda(I - h\lambda D)^{-1}(A - D)) < 1$, waarin $\rho(\cdot)$ de spectrale-radius-functie voorstelt.

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. *Applied Numerical Mathematics*, 18:397–411, 1995.

2. Door in het PILSRK proces gepresenteerd in Hoofdstuk 7 van dit proefschrift de parameters s, γ, β en δ respectievelijk 2, 1, 0 en 0 te kiezen, met Radau IIA als onderliggende Runge–Kutta methode, ontstaat een transformatievrij numeriek schema

van derde orde dat geschikt is voor implementatie op een sequentiële computer omdat de twee lineaire stelsels die per iteratie opgelost moeten worden dezelfde matrix hebben.

J.J.B. DE SWART. A simple ODE solver based on 2-stage Radau IIA. *Journal of Computational and Applied Mathematics*, 84(2):277–280, 1997.

3. Van de twee iteratieve methoden voor het oplossen van lineaire systemen met een niet-symmetrische matrix, de 'Quasi-Minimal Residual' methode (QMR) en de 'Bi-Conjugate Gradient' methode (BiCG), kan QMR weliswaar gladder convergeren dan BiCG, maar niet essentieel sneller.

   J.J.B. DE SWART. *The Quasi-Minimal Residual method.* Master's thesis, University of Utrecht, 1993.

4. Een bestaand computerprogramma voor het numeriek oplossen van differentiaalvergelijkingen is redelijk robuust als het alle problemen in [1] op kan lossen, maar pas ècht robuust als het ook de problemen op kan lossen die in de toekomst aan deze verzameling toegevoegd zullen worden, omdat het programma niet afgestemd is op deze nieuwe problemen.

   [1] 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. Available at `http://www.cwi.nl/cwi/projects/IVPtestset/`.

5. De hoeveelheid toeters en bellen bepalen de populariteit van numerieke software in grotere mate dan de kwaliteit van de onderliggende methode.

6. De kwaliteit van het referee-proces van tijdschriftartikelen kan vergroot worden door de namen van de referees op te nemen in het artikel, maar dan zou in veel gevallen de acceptatiedatum van een artikel geantidateerd moeten worden, teneinde de schande voor de referees te beperken.

7. Uit het feit dat een gebroken beschuit minder lekker is dan een hele, blijkt wel dat smaak psychisch is.

8. Idealen moeten zo groot gekozen worden, dat ze nog te zien zijn als men in een dal zit.

9. Aan alles wat met onderneemt kleven vele nadelen, die echter niet opwegen tegen het nadeel van niets ondernemen.

10. In plaats van te denken aan luchtvervuiling, is het ook mogelijk om de strepen die vliegtuigen in de lucht achterlaten te associëren met de vrijheid die het reizen per vliegtuig teweegbrengt.