UNIVERSITÀ DEGLI STUDI DI BARI
DOTTORATO DI RICERCA IN MATEMATICA
XVIII CICLO – A.A. 2004–2005
SETTORE SCIENTIFICO DISCIPLINARE:
ANALISI NUMERICA – MAT/08

TESI DI DOTTORATO

Numerical solution of Bordered Almost Block Diagonal linear systems arising from BVPs

Candidato:
Giuseppe ROMANAZZI

Supervisore della tesi:
Prof. Pierluigi AMODIO

Prof. Ian GLADWELL

Coordinatore del Dottorato di Ricerca:
Prof. Luciano LOPEZ

To Tanya, Marina,
and Kwam

Contents

Introduction **1**

1 Solution techniques for BVPs **7**
1.1 Conditioning in IVPs and BVPs 8
1.1.1 Well-conditioned problem 8
1.1.2 Theoretical solution of a problem 9
1.1.3 Uniform asymptotic stability and dichotomy 11
1.2 Difference methods 15
1.3 The Mono-Implicit Runge Kutta formulae 16
1.3.1 The code *MIRKDC* 16
1.4 Multiple shooting method 17
1.5 Splines 18
1.6 Spline collocation at Gaussian points 19
1.7 B-spline and monomial spline collocation 20
1.7.1 B-splines 21
1.7.2 Monomial splines 24

2 Algorithms for ABD and BABD systems **27**
2.1 The Wright example 27
2.2 Rewriting a BABD system as an ABD system 29
2.3 Gaussian elimination with partial pivoting 30
2.4 Alternating row and column elimination 31
2.5 Modified alternate row and column elimination 36
2.5.1 The *ABDPACK* algorithm 39
2.6 The *RSCALE* algorithm 40

3 An algorithm for the solution of BABD systems based on cyclic reduction **45**
3.1 The cyclic reduction algorithm 45
3.2 Computational cost and required memory 49
3.3 Stability 50
3.4 Solution of transposed BABD systems 51
3.5 Description of the software 55

3.6 Calling sequences . . . 58
3.6.1 A new version of *MIRKDC* . . . 60
3.7 Comparisons among the linear system solvers . . . 60
3.8 Comparisons of the two versions of *MIRKDC* . . . 62

4 Algorithms to solve general BABD linear systems 65
4.1 Introduction . . . 65
4.2 Cyclic reduction approach for general BABD linear systems . . . 68
4.3 An algorithm for BABD systems arising from OSC . . . 72
4.4 *GBABDCR*: an optimized implementation of the cyclic reduction . 74
4.4.1 Description of the software . . . 76
4.5 Numerical comparisons . . . 77
4.6 An approach to solve BABD systems in the monomial case . . . 80
4.6.1 The algorithm *BABDCR_MONO* . . . 80
4.6.2 Description of the software . . . 83
4.7 Comparisons in the monomial case . . . 85

5 A parallel algorithm for the solution of BABD systems using MPI 87
5.1 The parallel algorithm . . . 87
5.1.1 Factorization phase . . . 89
5.1.2 Solution phase . . . 90
5.1.3 A note about the startup . . . 90
5.2 Code in a pseudo programming language . . . 91
5.3 Speedup . . . 95

6 Appendix of Numerical Analysis 99
6.1 Polynomials . . . 99
6.2 Runge-Kutta formulae . . . 100
6.3 MIRK/CMIRK schemes in the code *MIRKDC* . . . 101

Acknowledgements 103

Bibliography 105

Introduction

We analyze the solution of Bordered Almost Block Diagonal linear systems using new algorithms which are introduced and described in detail. All these algorithms use a cyclic reduction approach with LU factorization. Cyclic reduction, despite other techniques, permits to obtain a parallelizable version of the code. Moreover, these algorithms are efficient even in serial version, we provide comparisons in timing and accuracy with respect to previous codes.

Almost Block Diagonal (ABD) [1, 18, 35] and Bordered Almost Block Diagonal (BABD) [1, 35] linear systems $A\mathbf{y} = \mathbf{f}$ arise in discretizations of Boundary Value Problems (BVPs) with, respectively, separated and non-separated boundary conditions, for ordinary and partial differential equations.

The ABD matrices associated to ABD linear systems are sparse matrices characterized by a very special pattern: the nonzero elements are grouped in block rows, there is no intersection between the nonzero columns of two non-consecutive block rows; finally, the main diagonal entries always lie inside the nonzero blocks. BABD matrices, associated to BABD linear systems, satisfy a further property, see Figure 1: the first (or the last) block row has an additional block in the right-upper (left-lower) corner whose nonzero columns only intersect some columns of the last (first) block row. In Figures 2 and 3, the most frequently occurring ABD and BABD structures arising in BVP solvers are shown. In these structures, the blocks S_i and R_i are square and have the same dimension.

In Chapter 1, we outline how BABD structures arise from Boundary Value Problem (BVP) techniques, as the Orthogonal Spline Collocation (OSC). In particular, we consider BVP for Ordinary Differential Equations (ODEs) with non-separated boundary conditions.

In Chapter 2, we describe some numerical techniques and approaches for solving ABD and BABD linear systems. These techniques are compared to the proposed algorithms in the successive chapters. A list of codes using some of these techniques are listed in the following two sections.

In Chapter 3, we introduce the new sequential BABD algorithm *BABDCR* [3, 4] which is based on the cyclic reduction. *BABDCR* has been written in Fortran 90. We compare it with respect to other codes on randomly generated systems and on systems arising from the BVP solver *MIRKDC* [34].

In Chapter 4 we concern with a general version of *BABDCR* in order to solve

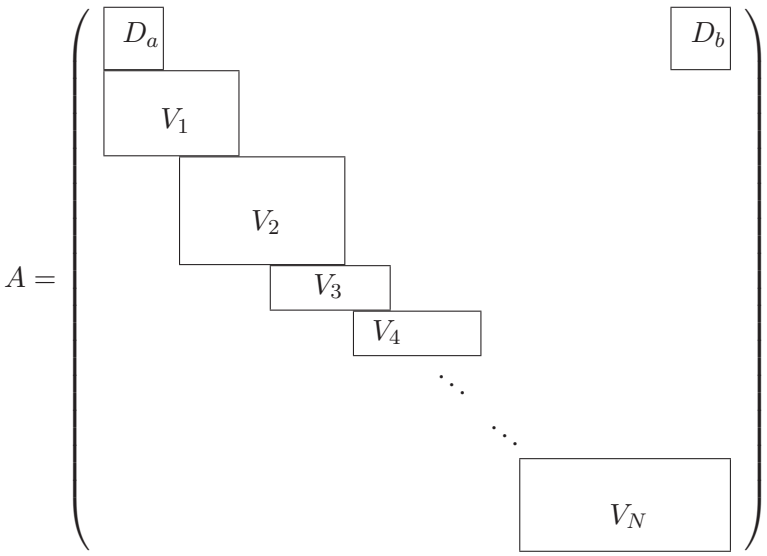

Figure 1: General BABD structure

$$\begin{pmatrix} D_{top} & & & & \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \\ & & & & D_{bot} \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d}_a \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \\ \mathbf{d}_b \end{pmatrix}$$

Figure 2: An ABD linear system structure, $D_{top} \in \mathbb{R}^{m_{top} \times m}$, $D_{bot} \in \mathbb{R}^{(m-m_{top}) \times m}$, S_i, $R_i \in \mathbb{R}^{m \times m}$, $\mathbf{y}_i \in \mathbb{R}^m$, $i = 0, \ldots, N$, $\mathbf{f}_j \in \mathbb{R}^m$, $j = 1, \ldots, N$, $\mathbf{d}_a \in \mathbb{R}^{m_{top}}$, $\mathbf{d}_b \in \mathbb{R}^{m-m_{top}}$.

$$\begin{pmatrix} D_a & & & & D_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}$$

Figure 3: A BABD linear system structure, D_a, D_b, S_i, $R_i \in \mathbb{R}^{m \times m}$,$\mathbf{y}_i \in \mathbb{R}^m$, $i = 0, \ldots, N$, $\mathbf{f}_j \in \mathbb{R}^m$, $j = 1, \ldots, N$, $\mathbf{d} \in \mathbb{R}^m$.

BABD systems with blocks of different dimensions. We present many approaches and, in particular, we employ an algorithm for solving general BABD systems arising from OSC, and an algorithm for solving only BABD systems with the particular structure arising from OSC with monomial spline basis. Comparisons between the new codes presented and the well-known *COLROW* [28, 29], based on a column-row elimination technique described in Chapter 2, are provided.

Finally, in Chapter 5, we analyze a parallel version of *BABDCR* on distributed memory parallel architecture using MPI (Message Passing Interface). This algorithm has been implemented in Fortran 90 on a parallel architecture with 16 processors. The speed-up of this parallelized code with respect to the serial version confirms the great advantage that the cyclic reduction techniques have with respect to sequential techniques, as the elimination techniques described in Chapter 2, in order to solve very large BABD (or ABD) linear systems.

Serial solvers for ABD systems

Because of its relevance in BVP codes, the solution of ABD systems has been the subject of long-term research, and several codes have been proposed. In [1] and, more recently in [35], a very rich literature on this subject may be found. The first code for ABD systems was *SOLVEBLOK* [20, 21]; as for standard LU factorization applied to banded systems, it requires fill-in to ensure stability. The alternate row and column stable elimination, called Varah's procedure [74], exploited the sparsity structure of the ABD matrices to avoid fill-in. This procedure is also denoted by ARCE in the following chapters. The packages *COLROW* and *ARCECO* in [28, 29] are based on a modified version of Varah's procedure (MARCE), which leads to a reduction in the number of arithmetic operations. *COLROW* solves ABD linear systems with row blocks of the same dimensions and *ARCECO* solves general ABD linear systems with blocks of varying dimensions. Numerical experiments in [29, 40] demonstrate their efficiency and their superiority with respect to *SOLVEBLOK*, that is, less storage and execution time on systems arising from BVODEs and on randomly generated linear systems. When *SOLVEBLOK* is applied to ABD linear systems arising from B-spline discretization, as in the BVODE code *COLSYS* [6, 7, 8], it causes fill-in. In [29] it is proved that *COLROW* solves these linear systems without fill-in and saving of up to 50% in execution time over *SOLVEBLOK*.

The NAG library [69] contains the routine *F01LHF* [22], a variant of *ARCECO* using level 2 BLAS routines [31], which solves both the systems: with an ABD coefficient matrix, and with its transpose. A level 3 BLAS [30] version of *ARCECO* has also been developed in [41, 42]; the corresponding code is available in [27]. Two variants of *COLROW* have been developed by Patrick Keast. The first *TRANSCOLROW* [50], as *F01LHF*, solves linear systems with an ABD coefficient matrix and with its transpose; the second variant, *COMPLEXCOLROW* [53], solves complex ABD linear systems and can be used in Padé methods applied to parabolic initial-

BVP using Orthogonal Spline Collocation (OSC) for the spatial discretization. The algorithm implemented in *LAMPAK* [55] refers to the Lam's method [59], there row and column interchanges are used to avoid fill-in but only row elimination is performed throughout. This technique has been applied in the analysis of beam structures [14] and its numerical stability is discussed in [73]. Some finite difference methods [37, 56] yield linear systems with an almost block diagonal structure where the overlap is between two consecutive column blocks. Code *ROWCOL* [36] use alternate row and column elimination to solve directly these kind of systems.
The solution of ABD systems arising from OSC with monomial spline basis [10], applied to a BVODE with separated boundary conditions, has also been considered. Since *COLROW* and *ARCECO*, as *SOLVEBLOK*, use fill-in to solve these systems, [63, 64] introduce the algorithm *ABDPACK*. It uses an alternate row and column elimination that, exploiting the special pattern of these systems, avoids the unnecessary fill-in. The same authors have implemented *ABBPACK* [63, 64], a special case of *ABDPACK*, which solves ABD systems with an upper block diagonal with identity matrices arising, for example, in implicit Runge-Kutta methods, in multiple shooting methods [1] and from condensation [10] applied to the original ABD system solved by *ABDPACK*.

Serial solvers for BABD systems

Due to the difficulty of treating boundary blocks, the solution of BABD systems has not received as much consideration. To our knowledge, the unique sequential algorithm designed specifically for BABD systems is the algorithm *BABDCR* [3, 4], described in Chapter 3. Other codes are written directly for parallel architectures, as *RSCALE* [49], that uses eigenvalue rescaling in order to ensure numerical stability for a wider class of BABD systems. In the sequential *PASVAR* codes [60, 61, 78], the routines *DECOMP* and *SOLVE* are used for the factorization and the back-substitution phase, respectively. These two routines have been written for ABD systems but they can handle even BABD systems. A detailed description of the various stable (and non) techniques for solving BABD systems is considered in [38]. One stable technique consists in writing the BABD system as ABD system of approximately twice the size, and then, in using ABD or banded matrix algorithms. This technique implies high computational cost due to doubling the dimension of the system and inefficiency, taking no account of the structure within the rewritten system. Alternative techniques consist in using algorithms for solving general bordered systems. As an example, *BEMW* [44] employes a recursive bordering technique to build up a solution of a bordered system from the solution of a much simpler system. Numerical tests [38, 79] prove the instability of such approach in some circumstances. Vice versa, a stable and efficient technique consists in using orthogonal factorization directly on the BABD matrix, storing Householder matrix in vector form to save memory. However, even this technique results in high computational cost.

ABD solvers in ODE and PDE packages

There are nonlinear boundary value ODE (BVODE) packages and nonlinear initial-boundary value PDE packages that employ the solution of ABD systems.

As BVODE package, *COLSYS* [6, 7, 8] uses *SOLVEBLOK* for solving the ABD linear systems which arise from the application of the Orthogonal Spline Collocation (OSC) to the given BVODE at Gaussian points with a B-spline basis. *COLNEW* [13] and *PCOLNEW* [16] uses modified versions of *SOLVEBLOK* for solving the ABD linear systems which arise from OSC applied to the BVODE at Gaussian points with a monomial spline basis. Its NAG version, *D02TKF*, uses the ABD solver code *F01LHF*. If applied to linear systems generated from OSC with monomial basis, *COLROW* and *SOLVEBLOK* generate fill-in while ABDPACK no requires additional storage.

ABD linear systems can be solved in two manner. The first is to solve it directly using some variant of Gaussian elimination, the second is to condense it, in the sense described in [10], and then to solve directly a smallest ABD linear system with the common structure in Figure 2. In *COLNEW* it is implemented this last process, it uses codes from *LINPACK* [32] to perform the condensation and *SOLVEBLOK* to solve the condensed linear systems. If the condensed linear system is solved using *ABBPACK* no significant savings are obtained. It is preferable to use the condensation process on shared memory machines for its high parallel performance with respect to the direct solution. We discuss the condensation process for the serial code *GBABDCR* in Section 4.4.

The Mono Implicit Runge Kutta (MIRK) code with defect control *MIRKDC* [33, 34] uses the ABD solver *COLROW*. A modified version of *MIRKDC* is presented in Section 3.6.1, it handles BVODE problems with non-separated boundary condition using the BABD solver *BABDCR*.

Three BVODE packages use modified versions of *COLROW*: *TWPBVP* [26], a deferred correction code; *COLMOD* [75], a modified version of *COLNEW*, and *ACDC* [25] that uses automatic continuation and OSC at Lobatto points to solve singularly perturbed BVODEs.

All the PDEs packages using ABD codes, solve systems of initial-boundary value PDEs in one space variable. They use a method of lines (MOL) where the OSC is used for the discretization of the spatial variable.

The codes *PDECOL* [62] and its variant *EPDCOL* [57], both using B-spline as basis functions for the spline collocation, solve the ABD systems resulting from the collocation, with *SOLVEBLOK* and *COLROW*, respectively. A MOL code, cited in [70], is based on spline collocation with monomial bases and uses *ABDPACK* for solving the resulting ABD systems. See [35] for a complete list of PDE packages using ABD solvers.

Chapter 1

Solution techniques for BVPs

We describe solution techniques for Two Point Boundary Value Problems which yield BABD linear systems. In the first section, we introduce the concept of well conditioning of Initial and Boundary Value Problems for Ordinary Differential Equations and we analyze necessary and sufficient conditions which guarantee well conditioning of the problem.

BABD linear systems arise from using various techniques, like Multiple Shooting [9, 48, 58, 71] , and difference methods [1, 9, 23, 58], applied to a BVP for ODEs. A brief description of these approaches is given. We analyze, particularly, the Orthogonal Spline Collocation (OSC) method as an effective technique for approximating the solution of boundary value problems for ODE. This technique is also exceedingly efficient to approximate the solution of boundary value problems with time dependent PDEs, using the method of lines [70]. The BABD structure arising depends on the spline basis used. B-spline and monomial spline basis [10, 18] cases are considered.

All the above ODE techniques can be applied to a general mth order nonlinear ODE

$$D^m u(x) = F(x, \mathbf{z}(u(x))), \qquad u(x) \in \mathbb{R},\ x \in [a, b], \tag{1.1}$$

with the boundary conditions

$$\mathbf{g}(\mathbf{z}(u(a)), \mathbf{z}(u(b))) = \mathbf{0}, \tag{1.2}$$

where

$$\mathbf{z}(u(x)) := (u(x), Du(x), D^2u(x), \ldots, D^{m-1}u(x))^T,$$
$$F(x, \mathbf{z}) : [a, b] \times \mathbb{R}^m \to \mathbb{R},$$

and

$$\mathbf{g}(\mathbf{z}, \mathbf{w}) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m.$$

While OSC is applied directly to (1.1)-(1.2), in the case of Multiple Shooting or Finite Difference Methods, it is convenient to consider an equivalent model

$$\begin{aligned} &\mathbf{y}' = \mathbf{f}(x, \mathbf{y}(x)), \\ &\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}, \end{aligned} \tag{1.3}$$

where

$$\mathbf{y}(x) = (y_1(x), \ldots, y_m(x))^T \in \mathbb{R}^m, \qquad y_i(x) = D^{i-1}u(x),$$

that is $\mathbf{y}(x) = \mathbf{z}(u(x))$.

Using the above cited techniques, Boundary Value Problems (BVPs) with non-separated boundary conditions, as in (1.3), produce BABD linear systems, whereas BVPs with separated boundary conditions,

$$B_a\mathbf{y}(a) = \mathbf{d}_a, \qquad B_b\mathbf{y}(b) = \mathbf{d}_b,$$

yield linear systems which have the ABD form shown in Figure 2.

For what concerns the numerical solution (see from Section 1.2 to the end of chapter), we can reduce the nonlinear problem (1.1)-(1.2), using a quasilinearization process [15, 71], to a sequence of mth order linear differential equations

$$\begin{aligned} &L(u)(x) := D^m u(x) - \sum_{l=1}^{m} c_l(x) D^{l-1}u(x) = f(x), \\ &B_a\mathbf{z}(u(a)) + B_b\mathbf{z}(u(b)) = \mathbf{d}. \end{aligned} \tag{1.4}$$

Moreover, quasilinearization approaches [1] reduce any first order nonlinear system (1.3) to a sequence of linear differential systems in the form

$$\begin{aligned} &\mathbf{y}' = A(x)\mathbf{y}(x) + \mathbf{q}(x), \\ &B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{d}. \end{aligned} \tag{1.5}$$

Thus in the next sections, we describe most of the discretization techniques for BVODEs in the linear form (1.4) or (1.5), this allows also a consistent analysis of the well-conditioning of the problem.

1.1 Conditioning in IVPs and BVPs

We discuss the conditioning of linear Initial and Boundary Value Problems for Ordinary Differential Equations in order to begin the numerical approximation.

1.1.1 Well-conditioned problem

We consider Initial Value Problem (IVP) and Boundary Value Problem (BVP) for ODEs, respectively, in the form

$$\begin{aligned} &\mathbf{y}' = A(x)\mathbf{y}(x) + \mathbf{q}(x), \qquad a \le x \le b, \\ &\mathbf{y}(a) = \mathbf{d}_a, \end{aligned} \tag{1.6}$$

[1] An approach is the Quasi-Linearization Method (QLM) that applied to (1.3) generates the sequence of linear BVODE

$$\mathbf{y}'_{n+1}(x) = \mathbf{f}(x, \mathbf{y}_n(x)) + (\mathbf{y}_{n+1}(x) - \mathbf{y}_n(x))\frac{\partial f}{\partial \mathbf{y}}(x, \mathbf{y}_n(x)),$$

using the Newton iteration applied to $F(\mathbf{y}) = \mathbf{y}' - \mathbf{f}(x, \mathbf{y})$.

and

$$\begin{array}{l} \mathbf{y}' = A(x)\mathbf{y}(x) + \mathbf{q}(x), \qquad a \le x \le b, \\ B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{d}, \end{array} \tag{1.7}$$

where $\mathbf{y} : [a,b] \longrightarrow \mathbb{R}^n,\ A(x), B_a, B_b \in \mathbb{R}^{n\times n}, \mathbf{q}(x), \mathbf{d}, \mathbf{d}_a \in \mathbb{R}^n$.

Definition 1.1.1. Considered as a BVP or an IVP, a problem is *well-conditioned*, if 'small' changes in the data produce 'small' changes in the solution and it is *ill-conditioned* otherwise.

1.1.2 Theoretical solution of a problem

In order to express analytically the solution of an ODE system with initial and boundary conditions, we introduce the basic concept of *fundamental solution*. We suppose that the functions $A(x)$ and $\mathbf{q}(x)$ in (1.6) are continuous, which guarantees the existence and uniqueness of the solution in $[a,b]$. In fact, the continuity of $A(x)$ and $\mathbf{q}(x)$ yields $\mathbf{f}(x,\mathbf{y}(x)) = A(x)\mathbf{y}(x) + \mathbf{q}(x)$ is continuous and satisfies the Lipschitz conditions on $\mathbf{y}$: given a vector norm $\|.\|$ an $M > 0$ exists such that for any $\mathbf{y_1}, \mathbf{y_2} \in \mathbb{R}^n,\ x \in [a,b], i = 1,\ldots,n$

$$|f_i(x,\mathbf{y_1}) - f_i(x,\mathbf{y_2})| \le M\|\mathbf{y_1} - \mathbf{y_2}\|,$$

where $\mathbf{f}(x,\mathbf{y}) = (f_1(x,\mathbf{y}),\ldots,f_n(x,\mathbf{y}))$, hence, see [72], existence and uniqueness of the solution are proved.

Definition 1.1.2. A *fundamental solution* of problem (1.6) (or (1.7)) is any matrix function $Y(x)$ solution of the homogeneous ODE

$$\mathbf{y}' = A(x)\mathbf{y},$$

such that, for any $x \in [a,b]$, $Y(x) \in \mathbb{R}^{n\times n}$ has n linear independent columns.

Therefore any fundamental solution $Y(x)$ satisfies $Y'(x) = A(x)Y(x)$. If $\Phi(x)$ is another fundamental solution, a constant nonsingular $n\times n$ matrix R exists satisfying $\Phi(x) = Y(x)R$. In particular, if $\Phi(x)$ denotes the unique fundamental solution satisfying $\Phi(a) = I$, then $R = (Y(a))^{-1}$ and $\Phi(x) = e^{\int_a^x A(s)ds}$. Then in the constant coefficient case $A(x) = A$, we get $\Phi(x) = e^{(x-a)A}$ [2].

Definition 1.1.3. A *particular solution* $\mathbf{y}_p(x)$ of problem (1.6) (or (1.7)) satisfies $\mathbf{y}_p' = A(x)\mathbf{y}_p(x) + \mathbf{q}(x),\ \mathbf{y}_p(a) = 0$.

[2] e^{tA} with $t \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, denotes the matrix exponential $e^{tA} = UDU^{-1}$ where $U = [\mathbf{u_1},\ldots,\mathbf{u_n}]$ is the eigenvector matrix associated to A and

$$D = \begin{pmatrix} e^{\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & e^{\lambda_n t} \end{pmatrix},$$

where $\lambda_1,\ldots,\lambda_n$ are the eigenvalues of A.

Proposition 1.1.4. *Suppose that A(x) and* $\mathbf{q}(x)$ *are continuous, the unique solution of the Initial Value Problem* (1.6) *can be expressed in the form*

$$\mathbf{y}(x) = \Phi(x)\mathbf{d}_a + \Phi(x)\int_a^x \Phi^{-1}(t)\mathbf{q}(t)dt.$$

The existence of the unique solution of the BVP (1.7) is more difficult to obtain. In fact, it is easy to prove that, considering the IVP

$$\begin{array}{l} \mathbf{w}' = A(x)\mathbf{w} + \mathbf{q}(x), \qquad a \le x \le b, \\ \mathbf{w}(a) = \mathbf{s}, \end{array} \tag{1.8}$$

the BVP

$$\begin{array}{l} \mathbf{w}' = A(x)\mathbf{w} + \mathbf{q}(x), \qquad a \le x \le b, \\ B_a\mathbf{w}(a) + B_b\mathbf{w}(b) = \mathbf{d}, \end{array} \tag{1.9}$$

admits as many solutions $\mathbf{w}(x)$ as the number of zeros of the equation

$$B_a\mathbf{s} + B_b\mathbf{w}(b) = \mathbf{d}. \tag{1.10}$$

That is, each solution of the BVP (1.9) is the solution $\mathbf{w}(x)$ of the IVP (1.8) with $\mathbf{s} = \mathbf{w}(a)$ that satisfies (1.10), where B_a can be singular. This approach is used in the Shooting methods presented in Section 1.4. We can express the unique solution (if it exists), by means of the following

Proposition 1.1.5. *A solution* $\mathbf{y}(x)$ *of the BVP* (1.7) *is unique if and only if, given a fundamental solution* $Y(x)$ *of the BVP, the matrix* $Q = B_aY(a) + B_bY(b)$ *is nonsingular. Moreover the solution is*

$$\mathbf{y}(x) = Y(x)Q^{-1}\left(\mathbf{d} - B_bY(b)\int_a^b Y^{-1}(t)\mathbf{q}(t)dt\right) + Y(x)\int_a^x Y^{-1}(t)\mathbf{q}(t)dt.$$

If $\tilde{\Phi}(x)$ is the fundamental solution such that $B_a\tilde{\Phi}(a) + B_b\tilde{\Phi}(b) = I$, we have

$$\mathbf{y}(x) = \tilde{\Phi}(x)\mathbf{d} + \int_a^b G(x,t)\mathbf{q}(t)dt, \tag{1.11}$$

where

$$G(x,t) = \begin{cases} \tilde{\Phi}(x)B_a\tilde{\Phi}(a)\tilde{\Phi}^{-1}(t), & t \le x, \\ -\tilde{\Phi}(x)B_b\tilde{\Phi}(b)\tilde{\Phi}^{-1}(t), & t > x. \end{cases}$$

If $\Phi(x)$ is the fundamental solution such that $\Phi(a) = I$ and $Q \equiv B_a\Phi(a) + B_b\Phi(b)$ then

$$\mathbf{y}(x) = \Phi(x)Q^{-1}\mathbf{d} + \int_a^b G(x,t)\mathbf{q}(t)dt, \tag{1.12}$$

with

$$G(x,t) = \begin{cases} \Phi(x)Q^{-1}B_a\Phi(a)\Phi^{-1}(t), & t \le x, \\ -\Phi(x)Q^{-1}B_b\Phi(b)\Phi^{-1}(t), & t > x. \end{cases}$$

Let $p \in \mathbb{N}$, $\mathbf{u} : [a,b] \to \mathbb{R}^n$ we define $\|\mathbf{u}\|_p \equiv \left(\int_a^b \|\mathbf{u}(t)\|_2^p dt\right)^{\frac{1}{p}}$. The equation (1.12) allows us to determine that any solution $\mathbf{y}$ of the BVP (1.7) satisfies

Proposition 1.1.6.

$$\|\mathbf{y}\|_\infty \leq k_1\|\mathbf{d}\|_\infty + k_2\|\mathbf{q}\|_p, \tag{1.13}$$

where $\frac{1}{p} + \frac{1}{q} = 1$, $k_1 = \|Y(x)Q^{-1}\|_\infty$ *and* $k_2 = \max\limits_{a\leq x\leq b} \|G(x,\cdot)\|_q$.

Proof. Given $p, q \in \mathbb{N}$, for the Cauchy-Schwartz inequality we have

$$\max_{a\leq x\leq b} \int_a^b \|G(x,t)\mathbf{q}(t)\|_2 dt \leq \max_{a\leq x\leq b} \|G(x,\cdot)\|_q \|\mathbf{q}\|_p;$$

then from (1.12),

$$\|\mathbf{y}\|_\infty \leq \|Y(x)Q^{-1}\|_\infty \|\mathbf{d}\|_\infty + \|G(x,\cdot)\|_q \|\mathbf{q}\|_p.$$

□

This inequality will be used to prove in the next paragraph the well-conditioning of the problem.

1.1.3 Uniform asymptotic stability and dichotomy

In the case of the Initial Value Problem (1.6) we have a well-conditioned problem with $\|\mathbf{q}\|$ limited if and only if the null solution of the homogeneous equation $\mathbf{y}' = A(x)\mathbf{y}$ is uniformly asymptotically stable, that is for any $\epsilon > 0$ and $x_0 \in [a,b]$ there exists $\delta = \delta(\epsilon) > 0$ such that for all $\mathbf{y_0} \in B_\delta = \{\mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{y}\| < \delta\}$ the solution $\mathbf{y}(x)$ of the homogeneous equation, with $\mathbf{y}(x_0) = \mathbf{y_0}$, satisfies $\mathbf{y}(x) \in B_\epsilon$ with $x \geq x_0$.

In fact, let $\tilde{\mathbf{y}}$ the solution of the perturbed IVP

$$\begin{aligned} &\mathbf{z}' = (A(x) + \delta A(x))\mathbf{z} + \mathbf{q}(x) + \delta\mathbf{q}(x), \\ &\mathbf{z}(a) = \mathbf{d}_a + \delta\mathbf{d}_a, \end{aligned}$$

with $\delta A(x)$, $\delta\mathbf{q}(x)$ and $\delta\mathbf{d}_a$ small perturbations in norm, and let $\mathbf{y}(x)$ be the solution of the non-perturbed IVP, we have that $\mathbf{e}(x) = \tilde{\mathbf{y}}(x) - \mathbf{y}(x)$ satisfies the equation $\mathbf{e}'(x) = A(x)\mathbf{e}(x) + \delta\mathbf{q}(x) + \delta A(x)\mathbf{z}(x)$, with $\mathbf{e}(a) = \delta\mathbf{d}_a$; thus, for the Proposition 1.1.4, it follows that

$$\mathbf{e}(x) = \Phi(x)\delta\mathbf{d}_a + \int_a^x \Phi(x)\Phi(t)^{-1}(\delta\mathbf{q}(t) + \delta A(t)\mathbf{z}(t))dt.$$

Since the null solution is uniformly and asymptotically stable, it can be proven that the norm $\|\Phi(x)(\Phi(t))^{-1}\|$tends to zero as $x \to +\infty$ for all $t \geq a$, see [9]. For example, if $A(x) = A$ is a constant matrix, it implies that $\|\Phi(x)(\Phi(t))^{-1}\| = \|e^{(x-t)A}\|$ and if A has eigenvalues λ with $Re(\lambda) < 0$, then $\mathbf{e}(x)$ is bounded by a small perturbation on data and in particular $\mathbf{e}(x) \to 0$ for $x \to +\infty$.

Let us consider well-conditioning in the case of BVP (1.7). The exponential dichotomy for a BVP plays the same role of uniform asymptotic stability for an IVP; that is, exponential dichotomy is equivalent to well-conditioning.

Definition 1.1.7. Let $Y(x)$ be a fundamental solution for the ODE $\mathbf{y}' = A(x)\mathbf{y}$, with $A(x)$ continuous in $[a, b]$. The ODE has an *exponential dichotomy* if there exists a constant orthogonal projection matrix P (that is P is an orthogonal matrix and $P^2 = P$) of rank p, $0 \leq p \leq n$ and $K, \lambda, \mu > 0$ with K not too large such that

$$\begin{array}{ll} \|Y(x)PY^{-1}(t)\| \leq Ke^{-\lambda(x-t)}, & x \geq t, \\ \|Y(x)(I-P)Y^{-1}(t)\| \leq Ke^{-\mu(t-x)}, & x < t, \end{array} \tag{1.14}$$

for any $a \leq x, t \leq b$.

We observe that exponential dichotomy, if exists, it is satisfied for any fundamental solution. This property allows to separate the space of solutions of the homogeneous ODE into two subspaces, one containing the decreasing solutions (decreasing modes) and the other containing the increasing solutions (increasing modes). Indeed, given a fundamental solution $Y(x)$ of the ODE, if

$$S = \{Y(x)\mathbf{c}|\ \mathbf{c} \in \mathbb{R}^n\}$$

is the set of the solutions then

$$S = S_1 + S_2,$$

where

$$S_1 = \{Y(x)P\mathbf{c}|\ \mathbf{c} \in \mathbb{R}^n\},$$

and

$$S_2 = \{Y(x)(I-P)\mathbf{c}|\ \mathbf{c} \in \mathbb{R}^n\};$$

for the exponential dichotomy, it follows that for any $\mathbf{u}(x) \in S_1$ and $\mathbf{w}(x) \in S_2$ we get [3]

$$\begin{array}{ll} \dfrac{\|\mathbf{u}(x)\|}{\|\mathbf{u}(t)\|} \leq Ke^{-\lambda(x-t)}, & x \geq t \quad (\mathbf{u}(x)\ is\ a\ decreasing\ mode), \\ \dfrac{\|\mathbf{w}(x)\|}{\|\mathbf{w}(t)\|} \leq Ke^{-\mu(t-x)}, & x < t \quad (\mathbf{w}(x)\ is\ an\ increasing\ mode). \end{array}$$

It is possible determine a fundamental solution with the decreasing modes separated by the increasing modes.

Proposition 1.1.8. *If the ODE has an exponential dichotomy with $0 \leq p \leq n$, then there exists a fundamental solution $Z(x) = (Z_1(x)\ Z_2(x)) \in \mathbb{R}^{n\times n}$ of the ODE with $Z_1(x) \in \mathbb{R}^{n\times p}$ and $Z_2(x) \in \mathbb{R}^{n\times(n-p)}$ such that*

$$\begin{array}{ll} \|Z_1(x)PZ_1^{-1}(t)\| \leq Ke^{-\lambda(x-t)}, & x \geq t, \\ \|Z_2(x)(I-P)Z_2^{-1}(t)\| \leq Ke^{-\mu(t-x)}, & x < t, \end{array}$$

[3] Since $P^2 = P$ we have,

$$\frac{\|\mathbf{u}(x)\|}{\|\mathbf{u}(t)\|} = \frac{\|Y(x)PPc\|}{\|Y(t)Pc\|} = \frac{\|Y(x)PY^{-1}(t)Y(t)Pc\|}{\|Y(t)Pc\|} \leq \|Y(x)PY^{-1}(t)\| \leq Ke^{-\lambda(x-t)}.$$

The second inequality follows in a similar manner because $(I-P)^2 = (I-P)$.

with $P = \begin{pmatrix} 0 & 0 \\ 0 & I_p \end{pmatrix}$ *where* I_p *is the* $p \times p$ *identity matrix.*

In the constant coefficient case, $A(x) = A$, we can characterize the exponential dichotomy as a function of the eigenvalues of A.

Proposition 1.1.9. *The ODE* $\mathbf{y}' = A\mathbf{y}$ *has an exponential dichotomy if and only if* A *has eigenvalues with nonnull real parts. Moreover the orthogonal projection* $P = \begin{pmatrix} 0 & 0 \\ 0 & I_p \end{pmatrix}$ *is of rank* p *where*

$$p = card\{\lambda \in \mathbb{C} | Re(\lambda) < 0\} \quad \text{and} \quad n - p = card\{\lambda \in \mathbb{C} | Re(\lambda) > 0\}.$$

Now we prove that, for any BVP, exponential dichotomy of the homogeneous equation implies well-conditioning of the problem.

Theorem 1.1.10. *If the homogeneous equation* $\mathbf{y}' = A(x)\mathbf{y}$ *has an exponential dichotomy, then the corresponding BVP (1.7) is well-conditioned.*

Proof. Let K be the positive bounded constant used in Definition 1.1.7 concerning the exponential dichotomy and $k_1 = \|Y(x)Q^{-1}\|_\infty$, $k_2 = \max\limits_{a \le x \le b} \|G(x,\cdot)\|_\infty$ be the constants used in Proposition 1.1.6, for $q = +\infty$. We first prove that

$$k_2 \le K(2k_1 + 1).$$

For any $x \le t$, the function $G(x,t)$ and the fundamental solution $\Phi(x)$ in (1.12) satisfy

$$\begin{aligned} \|G(x,t)\| &= \|\Phi(x)Q^{-1}B_b\Phi(b)\Phi^{-1}(t)\| \\ &\le \|\Phi(x)Q^{-1}B_b\Phi(b)P\Phi^{-1}(t)\| \\ &\quad + \|\Phi(x)Q^{-1}B_b\Phi(b)(I-P)\Phi^{-1}(t)\|, \end{aligned}$$

where P is the orthogonal projection matrix of rank p. Without any loss of generality, we suppose that $\max\{\|B_a\|, \|B_b\|\} = 1$. Since $Q^{-1}B_b\Phi(b) = I - Q^{-1}B_a\Phi(a)$ and $\|\Phi(x)P\Phi^{-1}(t)\| \le Ke^{-\lambda(x-t)}$, it follows that

$$\begin{aligned} \|G(x,t)\| &\le K\|\Phi(x)Q^{-1}\|(\|B_b\|e^{-\lambda(b-t)} + \|B_a\|e^{-\mu(t-a)}) + Ke^{-\mu(t-x)} \\ &\le k_1K(e^{-\lambda(b-t)} + e^{-\mu(t-a)}) + Ke^{-\mu(t-x)}. \end{aligned}$$

Alternatively, since $x > t$ then

$$\|G(x,t)\|_\infty \le k_1K(e^{-\lambda(b-t)} + e^{-\mu(t-a)}) + Ke^{-\lambda(x-t)},$$

then $k_2 = \max\limits_{a \le x,t \le b} \|G(x,t)\|_\infty \le K(2k_1 + 1)$. Therefore, for the inequality (1.13) with $q = +\infty$ we get

$$\|\mathbf{y}\|_\infty \le k_1\|\mathbf{d}\|_\infty + K(2k_1 + 1)\|\mathbf{q}\|_\infty,$$

and the BVP is well-conditioned. □

Conversely, if we suppose that the BVP is well-conditioned, it has a dichotomy which results from the boundary conditions. For example, in the case of separated boundary conditions

$$\begin{aligned} &\mathbf{y}' = A(x)\mathbf{y} + \mathbf{q}(x), \\ &B_a\mathbf{y}(a) = \mathbf{d}_a,\ B_b\mathbf{y}(b) = \mathbf{d}_b, \end{aligned}$$

with $B_a \in \mathbb{R}^{p\times n}$, $B_b \in \mathbb{R}^{(n-p)\times n}$ the problem has an exponential dichotomy with an orthogonal projection of rank p.

Definition 1.1.11. Let $C, D \in \mathbb{R}^{m\times m}$, we denote with $\mathrm{rank[C|D]}$ the rank of the matrix $\begin{pmatrix} C & D \end{pmatrix}$ of size $m \times 2m$.

In the BVP case (1.7), the next proposition is satisfied

Proposition 1.1.12. *If* $\mathrm{rank[B_a|B_b]} = \mathrm{n}$*, then there exist* $p \in \mathbb{N}$, $0 \le p \le n$ *and a nonsingular matrix* S *such that* $B_a = SD_1Q_1$ *and* $B_b = SD_2Q_2$*, where* Q_1, Q_2 *are orthogonal matrices, and*

$$D_1 = \begin{pmatrix} D_{11} & 0 \\ 0 & I_p \end{pmatrix}, \qquad D_2 = \begin{pmatrix} I_{n-p} & 0 \\ 0 & D_{22} \end{pmatrix},$$

with $D_{11} \in \mathbb{R}^{(n-p)\times(n-p)}$, $D_{22} \in \mathbb{R}^{p\times p}$ *containing only elements* 0 *or* 1.

The hypothesis of unique solution of the BVP requires that

$$\mathrm{rank[B_a|B_b]} = \mathrm{n}.$$

In fact, if the BVP (1.7) has a unique solution, from proposition 1.1.5 it follows that $\mathrm{rank(Q)} = \mathrm{n}$, and therefore, since $\mathrm{rank(A+B)} \le \mathrm{rank(A)} + \mathrm{rank(B)}$, if we suppose $\mathrm{rank[B_a|B_b]} < \mathrm{n}$ we get $\mathrm{rank(B_a)} + \mathrm{rank(B_b)} \le \mathrm{rank[B_a|B_b]} < \mathrm{n}$ and

$$n = \mathrm{rank(Q)} \le \mathrm{rank(B_aY(a))} + \mathrm{rank(B_bY(b))} = \mathrm{rank(B_a)} + \mathrm{rank(B_b)} < \mathrm{n}.$$

Theorem 1.1.13. *If the BVP (1.7) is well-conditioned and* $\mathrm{rank[B_a|B_b]} = \mathrm{n}$*, then it has an exponential dichotomy with projection matrix of rank* p*, where* p *is determined as in the previous Proposition.*

As a corollary, we have the following

Proposition 1.1.14. *Suppose that the BVP has boundary conditions* $B_{top}y(a) = d_a$ *and* $B_{bot}y(b) = d_b$*, where* $B_{top} \in \mathbb{R}^{m\times n}$ *and* $B_{bot} \in \mathbb{R}^{(n-m)\times n}$ *with* $\mathrm{rank(B_{top})} = \mathrm{m}$ *and* $\mathrm{rank(B_{bot})} = \mathrm{n} - \mathrm{m}$*. Then, well-conditioning implies an exponential dichotomy with rank* $p = m$*.*

Proof. Let $B_a = \begin{pmatrix} B_{top} \\ 0 \end{pmatrix}$ and $B_b = \begin{pmatrix} 0 \\ B_{bot} \end{pmatrix}$; then we have $B_ay(a) + B_by(b) = \begin{pmatrix} d_a \\ d_b \end{pmatrix}$, with $\mathrm{rank[B_a|B_b]} = \mathrm{n}$. Therefore, from the previous theorem, the problem has the exponential dichotomy with rank $p = m$. □

1.2 Difference methods

Linear multistep difference schemes

$$\sum_{l=0}^{k} \alpha_l y_{i+l} = h\Phi(x_i; y_i, \ldots, y_{i+k}; h)$$

can be used to get high order accuracy in the approximation of solutions of BVODEs. The most popular and simplest methods are the (implicit) Midpoint rule and the Trapezoidal rule which have both highest order (second) in the class of linear one-step difference schemes [45, 58]. We show that, considering the linear BVODE (1.5) on $[a, b]$, both schemes give BABD systems. Let

$$\Delta : a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b \tag{1.15}$$

be a partition of $[a, b]$ with stepsize $h_i = x_{i+1} - x_i$ and $x_{i+\frac{1}{2}} = \dfrac{x_i + x_{i+1}}{2}$. Using the (implicit) Midpoint rule we have

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i A(x_{i+\frac{1}{2}}) \frac{\mathbf{y}_i + \mathbf{y}_{i+1}}{2} + h_i \mathbf{q}(x_{i+\frac{1}{2}}), \qquad i = 0, \ldots, N-1,$$

that is, we obtain

$$S_i \mathbf{y}_i + R_{i+1} \mathbf{y}_{i+1} = \mathbf{f}_{i+1}, \qquad i = 0, \ldots, N-1,$$

with

$$S_i = -I - \tfrac{h_i}{2} A(x_{i+\frac{1}{2}}),\ R_{i+1} = I - \tfrac{h_i}{2} A(x_{i+\frac{1}{2}}), \qquad \text{and}$$
$$\mathbf{f}_{i+1} = h_i \mathbf{q}(x_{i+\frac{1}{2}}).$$

Using the trapezoidal rule, we have for $i = 0, \ldots, N-1$,

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h_i}{2}(A(x_{i+1})\mathbf{y}_{i+1} + \mathbf{q}(x_{i+1}) + A(x_i)\mathbf{y}_i + \mathbf{q}(x_i)),$$

that is, we obtain the linear equations

$$S_i \mathbf{y}_i + R_{i+1} \mathbf{y}_{i+1} = \mathbf{f}_{i+1}, \qquad i = 0, \ldots, N-1,$$

with

$$S_i = -I - \tfrac{h_i}{2} A(x_i),\ R_{i+1} = I - \tfrac{h_i}{2} A(x_{i+1}), \qquad \text{and}$$
$$\mathbf{f}_{i+1} = \frac{h_i}{2}(\mathbf{q}(x_{i+1}) + \mathbf{q}(x_i)).$$

In both cases, the linear system obtained has the following structure

$$\begin{pmatrix} B_a & & & & B_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}.$$

1.3 The Mono-Implicit Runge Kutta formulae

Runge Kutta schemes, presented in Appendix, have been considered for the numerical solution of IVODE. In particular some recent works have focused on a subclass of implicit RK schemes called Mono Implicit Runge Kutta (MIRK) schemes [23], see [66] for a complete reference, which are also used for the numerical solution of BVODE.

The Mono Implicit Runge Kutta formulae (MIRK) are determined choosing a Runge Kutta scheme with an ulterior coefficient vector $\mathbf{v} := \{v_r\}_{r=1}^{s}$ and the associated scheme is

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \sum_{r=1}^{s} b_r K_{i,r},$$

with the stages

$$\mathbf{K}_{i,r} = \mathbf{f}(t_i + c_r h_i, (1 - v_r)\mathbf{y}_i + v_r \mathbf{y}_{i+1} + h_i \sum_{j=1}^{r-1} x_{rj} \mathbf{K}_{i,j}), \qquad r = 1, \ldots, s. \tag{1.16}$$

Thus, the corresponding Butcher array is

$$\begin{array}{c|c|c} \mathbf{c} & \mathbf{v} & \mathbf{X} \\ \hline & & \mathbf{b} \end{array}. \tag{1.17}$$

In order to describe how BABD systems arise from MIRK formulae, we discuss their application in the code *MIRKDC*.

1.3.1 The code *MIRKDC*

The code [34] solves BVPs

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)), \qquad t \in [a, b], \tag{1.18}$$

where $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$, with separated BCs

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \begin{pmatrix} \mathbf{g}_0(\mathbf{y}(a)) \\ \mathbf{g}_1(\mathbf{y}(b)) \end{pmatrix} = 0, \tag{1.19}$$

where $\mathbf{g} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$, $\mathbf{g}_0 : \mathbb{R}^m \to \mathbb{R}^{m_0}$, $\mathbf{g}_1 : \mathbb{R}^m \to \mathbb{R}^{m_1}$ and $m_0 + m_1 = m$.

It uses MIRK formulae to discretize (1.18) on a given subdivision $\{t_i\}_{i=0}^{N}$ of $[a, b]$. A continuous solution approximation $\mathbf{u} = \mathbf{u}(t)$ is obtained using a Continuous MIRK (CMIRK) scheme [67], to provide defect control and mesh selection capabilitie, see the Appendix where the scheme MIRK/CMIRK employed in *MIRKDC* is described in detail.

The MIRK scheme yields the nonlinear system

$$\mathbf{\Phi}(\mathbf{Y}) = 0,$$

which is solved using the Newton iteration

$$\mathbf{Y}^{(q+1)} = \mathbf{Y}^{(q)} + \Delta\mathbf{Y}^{(q)}, \qquad q = 0, 1, \ldots.$$

where

$$\left[\frac{\partial \mathbf{\Phi}(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}}\right] \Delta\mathbf{Y}^{(q)} = -\mathbf{\Phi}(\mathbf{Y}^{(q)}), \tag{1.20}$$

given $\mathbf{Y}^{(0)}$.
If the boundary conditions are separated, the Jacobian matrix in (1.20) has the ABD form

$$\frac{\partial \Phi(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}} = \begin{pmatrix} B_{top} & & & & \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \\ & & & & B_{bot} \end{pmatrix}$$

with $B_{top} = \frac{d\mathbf{g}_0(\mathbf{y}_0^{(q)})}{d\mathbf{y}_0}$ and $B_{bot} = \frac{d\mathbf{g}_1(\mathbf{y}_N^{(q)})}{d\mathbf{y}_N}$, on the contrary, if the boundary conditions are non-separated, it has the BABD form

$$\frac{\partial \Phi(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}} = \begin{pmatrix} B_a & & & & B_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix}$$

with $B_a = \frac{\partial \mathbf{g}(\mathbf{y}_0^{(q)}, \mathbf{y}_N^{(q)})}{\partial \mathbf{y}_0}$ and $B_b = \frac{\partial \mathbf{g}_1(\mathbf{y}_0^{(q)}, \mathbf{y}_N^{(q)})}{\partial \mathbf{y}_N}$ and in both cases for $i = 0, \ldots, N-1$,

$$S_i = -I - h_i \sum_{r=1}^{s} b_r \left(\frac{\partial \mathbf{K}_{r,i}}{\partial \mathbf{y}_i}\right) \quad \text{and} \quad R_{i+1} = I - h_i \sum_{r=1}^{s} b_r \left(\frac{\partial \mathbf{K}_{r,i}}{\partial \mathbf{y}_{i+1}}\right).$$

MIRKDC employees *COLROW* to solve systems (1.20). The paralle version *PMIRKDC* [68] of the *MIRKDC* code uses the *RSCALE* algorithm for solving linear systems (1.20).

1.4 Multiple shooting method

Given a partition (1.15) of the time interval $[a, b]$ of the BVP (1.5), the multiple shooting method solves the BVP using a fundamental solution $Y_i(x) \in \mathbb{R}^{m\times m}$ and a particular solution $\mathbf{y}_{p,i}(x) \in \mathbb{R}^m$ (see definitions 1.1.2 and 1.1.3) on each

subinterval $[x_i, x_{i+1}]$, for $i = 0, \ldots, N-1$. Therefore, $Y_i(x)$ and $\mathbf{y}_{p,i}(x)$ satisfy, for any $x \in [x_i, x_{i+1}]$

$$\begin{aligned} Y_i' &= A(x)Y_i(x),\ Y_i(x_i) = I, \\ \mathbf{y}_{p,i}' &= A(x)\mathbf{y}_{p,i}(x) + \mathbf{q}(x),\ \mathbf{y}_{p,i}(x_i) = 0. \end{aligned}$$

Then, in order to obtain a solution of the BVP (1.5) in $[a, b]$, the method find $\mathbf{s}_i = \mathbf{y}(x_i) \in \mathbb{R}^m$ such that the solution

$$\mathbf{y}(x) = Y_i(x)\mathbf{s}_i + \mathbf{y}_{p,i}(x), \qquad x \in [x_i, x_{i+1}],$$

of the ODE

$$\mathbf{y}' = A(x)\mathbf{y}(x) + \mathbf{q}(x),$$

satisfies the boundary conditions

$$B_a\mathbf{s}_0 + B_b\mathbf{s}_N = \mathbf{d},$$

and preserves the continuity in the internal points $x_1, \ldots, x_{N-1}$, which means

$$\mathbf{s}_{i+1} = Y_i(x_{i+1})\mathbf{s}_i + y_{p,i}(x_{i+1}).$$

We obtain the following BABD linear system with $\{\mathbf{s}_i\}_{i=0}^N$ as unknowns

$$\begin{pmatrix} B_a & & & & & B_b \\ -Y_0(x_1) & I & & & & \\ & -Y_1(x_2) & I & & & \\ & & \ddots & \ddots & & \\ & & & -Y_{N-1}(x_N) & I & \end{pmatrix} \begin{pmatrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_{N-1} \\ \mathbf{s}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}, \tag{1.21}$$

with $\mathbf{f}_{i+1} := \mathbf{y}_{p,i}(x_{i+1})$, for $i = 0, \ldots, N-1$.

From the properties of fundamental solutions presented in Section 1.1.2, we obtain, in the constant coefficient case $A(x) = A$,

$$Y_i(x) = e^{(x-x_i)A}.$$

Hence, if $h_i = x_{i+1} - x_i$ is constant through the partition, a constant lower diagonal appears in (1.21) with

$$Y_i(x_{i+1}) = e^{h_i A}, \qquad i = 0, \ldots, N-1.$$

1.5 Splines

The approximation of a function $u(x)$, in a finite interval $[a, b]$, by an interpolating polynomial $p(x)$, gives an approximation error $\|u - p\|$, which can be controlled

by choosing the degree of the approximating polynomial sufficiently large and particular interpolating nodes, see [39]. In this case, the approximating function is generated by a polynomial basis.

Another way to approximate $u(x)$ is to use a function generated by a spline function basis. We start by considering the partition $\Delta = \{x_i\}_{i=0}^{N}$ in (1.15), upon the interval of approximation $[a, b]$ and by using low-degree polynomials on each subinterval $[x_i, x_{i+1}]$, called *splines*. Let $\mathbb{S}_{\Delta}^{k+m,m-1}$, also briefly denoted as $\mathbb{S}_{\Delta}$, be the *set of $\mathcal{C}^{m-1}$ splines of order $k+m$*, see appendix, defined on partition Δ

$$\mathbb{S}_{\Delta}^{k+m,m-1} := \left\{ s(x) \in \mathcal{C}^{m-1}[a,b] : s(x)|_{[x_i,x_{i+1}]} \in \mathcal{P}_{k+m}, i = 0, \ldots, N-1 \right\},$$

we find, in the approximation using splines, a function $s(x) \in \mathbb{S}_{\Delta}$ which reduces the approximation error $\|u - s\|$ by decreasing the *fineness* h of the partition Δ where

$$\begin{aligned} h &:= \max_{i=0,\ldots,N-1} h_i, \\ h_i &:= x_{i+1} - x_i, \ i = 0, \ldots, N-1. \end{aligned} \tag{1.22}$$

The expression of the approximation error depends on the choice of the spline basis. Splines, as piecewise polynomials, permit us to approximate a function with varying behavior over its domain, better than using a single polynomial. Moreover, since splines $s(x)$ have a certain number of continuous derivatives in $[a, b]$ (i.e. $m-1$, for $s(x) \in \mathbb{S}_{\Delta}$), this choice is preferable to other less smooth piecewise polynomials in the solution of Boundary Value Problems for ODEs of order m. We note that the dimension of $\mathbb{S}_{\Delta}$ is $kN+m$ [18], because the set of the piecewise polynomials, see appendix, $\mathcal{P}_{k+m,\Delta}$ (of order $k+m$) is of dimension $(k+m)N$ and we impose $m(N-1)$ continuity conditions on the internal points $x_1, \ldots, x_{N-1}$.

1.6 Spline collocation at Gaussian points

Collocation is a technique for approximating a function $u(x)$ on an interval $[a, b]$, where $u(x)$ is given (implicitly) as the solution of a general mth ordinary linear differential equation (1.4) with m non-separated boundary conditions.

It is assumed that, a sufficiently smooth, unique solution $u(x)$ exists and satisfies for some $c > 0$

$$\max_{a \le x \le b} |D^i u(x)| \le c, \qquad i = 0, \ldots, m.$$

Spline collocation, applied to (1.4), determines a spline approximation $u_{\Delta}(x) \in \mathbb{S}_{\Delta}^{k+m,m-1}$ to the exact solution $u(x)$ on the given mesh Δ, defined in (1.15), which satisfies the boundary conditions and the differential equation (1.4) on the given kN interior *collocation points*

$$\{\xi_{i,j}\}_{1 \le i \le N,\ 1 \le j \le k}.$$

That is, $u_\Delta(x)$ satisfies the *collocation equations*

$$L(u_\Delta)(\xi_{i,j}) = f(\xi_{i,j}), \qquad 1 \le i \le N \text{ and } 1 \le j \le k, \tag{1.23}$$

and

$$B_a\mathbf{z}(u_\Delta(a)) + B_b\mathbf{z}(u_\Delta(b)) = \mathbf{d}. \tag{1.24}$$

We choose the collocation equally distributed in each subinterval with k points per subinterval, that is, given

$$0 \le \rho_1 < \cdots < \rho_k \le 1, \tag{1.25}$$

we set the collocation points

$$\xi_{i,j} = x_{i-1} + \rho_j(x_i - x_{i-1}), \qquad 1 \le i \le N \text{ and } 1 \le j \le k. \tag{1.26}$$

We note that usually k is chosen such that $k \ge m$, see the examples in [10]. As proved in [19] and mentioned in [10, 18], if $\{\rho_j\}_{j=1}^k$ are chosen such that any polynomial $p(x) \in \mathcal{P}_n$, with $m \le n \le k$, satisfies

$$\int_0^1 p(x) \prod_{j=1}^{k} (x - \rho_j)dx = 0, \tag{1.27}$$

then $D^i u_\Delta(x)$ approximates $D^i u(x)$ to optimal order

$$\|D^i(u - u_\Delta)\| \le ch^{k+m-i}, \qquad 0 \le i \le m,$$

and the approximation is of order better than optimal order (super-convergent)

$$|D^i(u - u_\Delta)(x_j)| \le ch^{k+n}, \qquad 0 \le i \le m-1 \text{ and } 0 \le j \le N,$$

at the mesh points x_j, where h is the *fineness* of the partition Δ, defined in (1.22). Note that the zeros $\{\rho_i\}_{i=1,\ldots,k}$ of the kth orthogonal polynomial in $[0, 1]$, i.e. the kth Legendre polynomial, called *Gaussian points*, satisfy the orthogonality relation (1.27). Thus, if we choose $n = k$ and an orthogonal polynomial $p_n(x)$ in (1.27), we obtain the maximum order of convergence

$$|D^i(u - u_\Delta)(x_j)| \le ch^{2k}, \qquad 0 \le i \le m-1 \text{ and } 0 \le j \le N.$$

1.7 B-spline and monomial spline collocation

We derive the BABD structures which arise from spline collocation using a B-spline basis and a monomial spline basis, applied to the differential equation of order m on $[a, b]$ with non-separated boundary conditions (1.4). Given k Gaussian points (1.25), this approximation is defined by requiring:

(C1) $u_\Delta(x)$ is a polynomial of order $k + m$,

(C2) $u_\Delta(x) \in \mathcal{C}^{m-1}[a,b]$,

(C3) $u_\Delta(x)$ satisfies (1.23) at the kN collocation points (1.26),

(C4) $u_\Delta(x)$ satisfies the m boundary conditions (1.24).

Depending on the choice of the spline basis, these conditions impose a linear system of particular dimension and structure in the coefficients α_l, where $u_\Delta(x) = \sum \alpha_l \Phi_l(x)$ and the $\Phi_l(x)$ are basis functions.

1.7.1 B-splines

The choice of B-splines satisfies the demand to construct a basis for $\mathbb{S}_\Delta$ with the smallest possible support [65]. This is effective in approximation problems as surface fitting and curve design [18]. A possible basis $\{\Phi_{i,j}(x)\}$ has the *truncation power functions* [18, 65],

$$\Phi_{i,j}(x) := (x - x_{i-1})_+^j, \qquad i = 1,\ldots,N \text{ and } j = \mu_i,\ldots,k+m-1,$$

where

$$\mu_i = \begin{cases} 0 & \text{if } i = 1, \\ m & \text{elsewhere.} \end{cases}$$

Each $\Phi_{i,j}(x)$ has support $[x_{i-1}, +\infty[$. The B-spline basis $\{B_l(x)\}_{l=1}^{kN+m}$ of $\mathbb{S}_\Delta$ derives from the truncation power basis

$$B_l(x) := (t_{l+k+m} - t_l)[t_l, t_{l+1}, \ldots, t_{l+k+m}](\cdot - x)_+^{k+m-1}, \;\; l = 1,\ldots,kN+m$$

see Appendix, where $(\cdot - x)_+^{k+m-1}$ is a function f such that $f(y) = (y-x)_+^{k+m-1}$; $\{t_l\}_{l=1}^{(kN+m)+k+m}$ is a nondecreasing sequence which contains $(k+m)$ times $x_0 = a$, $x_N = b$, and k times each interior point x_1,...,x_{N-1}. This means

$$t_l := \begin{cases} x_0 & \text{if } l = 1,\ldots,k+m, \\ x_j & \text{if } l = jk+m+1,\ldots,(j+1)k+m, \text{ for } j = 1,\ldots,N-1, \\ x_N & \text{if } l = kN+m+1,\ldots,(k+1)N+2m. \end{cases}$$

We note that $B_l(x)$ has small support, $supp(B_l) = [t_l, t_{k+m+l}]$. Indeed, since the function $(\cdot - x)_+^{k+m-1}$ is a polynomial of order $k+m$, using a property of the $(k+m)$th divided difference, we have

$$[t_l, t_{l+1}, \ldots, t_{l+k+m}](\cdot - x)_+^{k+m-1} = 0.$$

It follows that in each interval $[t_l, t_{l+1}]$, for $l = 1,\ldots,N$, only $k+m$ B-splines $\{B_i(x)\}_{i=l-(k+m)+1}^{l}$ might be nonzero, which implies that in each interval $[x_i, x_{i+1}]$ only $k+m$ B-splines might be nonzero. Thus, let $u_\Delta(x) \in \mathbb{S}_\Delta$, then there exists some $\{y_l\}_{l=1}^{kN+m}$ [18] such that for any $i = 1,\ldots,N$

$$u_\Delta(x) = \sum_{l=(i-1)k+1}^{ik+m} y_l B_l(x), \qquad x \in [x_{i-1}, x_i]. \tag{1.28}$$

Concerning the collocation, since the spline space has dimension $kN+m$ and each combination

$$u_\Delta(x) = \sum_{l=1}^{kN+m} y_l B_l(x), \qquad x \in [a,b], \tag{1.29}$$

satisfies implicitly the continuity conditions (C2) in $[a,b]$, then each spline $u_\Delta(x)$ has $kN+m$ free parameters which can be determined by imposing the conditions (C3)-(C4). As shown in [18, 19], the equations associated to the conditions (C3)-(C4) can be solved using a Newton's iteration, which determines a sequence $\{u_{r,\Delta}(x)\}$ with

$$u_{r,\Delta}(x) = \sum_{l=1}^{kN+m} y_{r,l} B_l(x)$$

that converges to the solution $u_\Delta(x)$, for an initial guess $u_{0,\Delta}(x)$ sufficiently close to $u_\Delta(x)$. The Newton iteration is

$$\begin{aligned} &L(u_{r+1,\Delta})(\xi_{i,j}) = f(\xi_{i,j}) - \sum_{l=1}^{m} c_l(x) D^{l-1} u_{r,\Delta}(x), \\ &B_a \mathbf{z}(u_{r+1,\Delta}(a)) + B_b \mathbf{z}(u_{r+1,\Delta}(b)) = \mathbf{d}, \end{aligned} \tag{1.30}$$

where $i = 1, \ldots, N$ and $j = 1, \ldots, k$. An efficient implementation of B-splines in the collocation code *COLSYS*, described in [5, 8], imposes directly the conditions (C3)-(C4) on (1.29), that yields

$$\begin{aligned} &L(u_\Delta)(\xi_{i,j}) = \sum_{l=1}^{kN+m} y_l L(B_l)(\xi_{i,j}) = f(\xi_{i,j}), \quad 1 \le j \le k \text{ and } 1 \le i \le N, \\ &B_a \mathbf{z}(u_\Delta(a)) + B_b \mathbf{z}(u_\Delta(b)) = \mathbf{d}. \end{aligned} \tag{1.31}$$

Both the equations (1.30) and (1.31) correspond to a BABD linear system

$$A\mathbf{y} = \mathbf{f},$$

with the structure

$$A = \begin{pmatrix} D_a & & & & & & & & D_b \\ W_{1,1} & W_{1,2} & W_{1,3} & & & & & & \\ & & W_{2,1} & W_{2,2} & W_{2,3} & & & & \\ & & & & \ddots & & & & \\ & & & & & & & & \\ & & & & & & W_{N,1} & W_{N,2} & W_{N,3} \end{pmatrix}, \tag{1.32}$$

where for any $i = 1, \ldots, N$, the ith row block

$$V_i = (W_{i,1}\ W_{i,2}\ W_{i,3})$$

is of size $k \times (k+m)$, with $W_{i,1}, W_{i,3} \in \mathbb{R}^{k\times m}$ and $W_{i,2} \in \mathbb{R}^{k\times(k-m)}$. The ith row block corresponds to the k collocation equations (1.31) in $\xi_{i,j} \in [x_{i-1}, x_i]$, $j = 1, \ldots, k$. Since, $u_\Delta(x)|_{[x_{i-1},x_i]}$ depends only on the $k+m$ coefficients $\{y_l\}_{l=(i-1)k+1}^{ik+m}$, see (1.28), and the corresponding row block equation is

$$W_{i,1}\mathbf{z}_{i,1} + W_{i,2}\mathbf{z}_{i,2} + W_{i,3}\mathbf{z}_{i+1,1} = \mathbf{f}_i,$$

for $i = 1, \ldots, N$, where

$$\begin{aligned}\mathbf{z}_{i,1}^T &= \big(y_{(i-1)k+1}, \ldots, y_{(i-1)k+m}\big)^T \in \mathbb{R}^m,\\ \mathbf{z}_{i,2}^T &= \big(y_{(i-1)k+m+1}, \ldots, y_{ik}\big)^T \in \mathbb{R}^{k-m},\end{aligned}$$

and

$$\mathbf{f}^T = (\mathbf{d}, \mathbf{f}_1^T, \ldots, \mathbf{f}_N^T)^T,$$

with $\mathbf{f}_i^T = \big(\ f(\xi_{i,1}), f(\xi_{i,2}), \ldots, f(\xi_{i,k})\ \big)$. In *COLSYS*, this results in

$$W_{i,1} := (W_{i,1}^{j,r})_{1\le j\le k,\ 1\le r\le m}, \qquad W_{i,2} := (W_{i,2}^{j,r})_{1\le j\le k,\ 1\le r\le k-m},$$

$$\text{and } W_{i,3} := (W_{i,3}^{j,r})_{1\le j\le k,\ 1\le r\le m},$$

where for any $j = 1, \ldots, k$ and $r = 1, \ldots, m$,

$$W_{i,1}^{j,r} = L(B_{(i-1)k+r})(\xi_{i,j}), \qquad W_{i,3}^{j,r} = L(B_{ik+r})(\xi_{i,j}),$$

and for any $j = 1, \ldots, k$ and $r = 1, \ldots, k-m$,

$$W_{i,2}^{j,r} = L(B_{(i-1)k+m+r})(\xi_{i,j}).$$

Therefore, $W_{i,3}$ overlaps exactly the columns of $W_{i+1,1}$ corresponding to the unknowns $\mathbf{z}_{i+1,1}$. The blocks D_a and D_b in (1.32) correspond to the boundary conditions in (1.31),

$$D_a = B_a \begin{pmatrix} B_1(a) & \cdots & B_{k+m}(a) \\ DB_1(a) & \cdots & DB_{k+m}(a) \\ \vdots & & \vdots \\ D^{m-1}B_1(a) & \cdots & D^{m-1}B_{k+m}(a) \end{pmatrix}$$

and

$$D_b = B_b \begin{pmatrix} B_{(k-1)N+1}(b) & \cdots & B_{kN+m}(b) \\ DB_{(k-1)N+1}(b) & \cdots & DB_{kN+m}(b) \\ \vdots & & \vdots \\ D^{m-1}B_{(k-1)N+1}(b) & \cdots & D^{m-1}B_{kN+m}(b) \end{pmatrix}.$$

1.7.2 Monomial splines

Until the 1980s, the most popular approach in spline collocation was to use a B-spline basis, anyway recently in [10], it has been shown that the B-spline basis is inferior, in the computational cost and conditioning, with respect to a monomial basis when low continuity piecewise polynomials are used. We determine a monomial spline basis for the set $\mathbb{S}_\Delta$ using a monomial basis of the piecewise polynomials set $\mathcal{P}_{k+m}$. A monomial basis for $\mathcal{P}_{k+m}$ has $(k+m)N$ polynomials

$$\{\Phi_j(x-x_{i-1})\}_{1\le i\le N, 1\le j\le m}\,, \qquad \left\{h_i^m \Psi_j\left(\frac{x-x_{i-1}}{h_i}\right)\right\}_{1\le i\le N, 1\le j\le k},$$

each with support over only one subinterval $[x_{i-1}, x_i]$ with amplitude $h_i := x_i - x_{i-1},\ i = 1,\dots,N$. The first m polynomials are monomials locally

$$\Phi_j(x-x_{i-1}) = \frac{(x-x_{i-1})^{j-1}}{(j-1)!}, \qquad i = 1,\dots,N \ \text{ and } \ j = 1,\dots,m,$$

with each $\Phi_j(x)$ defined on $[0,1]$. The other k polynomials $\{\Psi_j(x)\}_{j=1}^k$ are chosen to satisfy the conditions

- $\Psi_j(x)$ is a polynomial of order $k+m$ defined on $[0,1]$,
- $\dfrac{d\Psi_j^{l-1}}{dx^{l-1}}(0) = 0, \qquad l = 1,\dots,m \ \text{ and } \ j = 1,\dots,k.$

One choice is to use monomials

$$\Psi_j(x) = \frac{x^{m+j-1}}{(m+j-1)!}, \qquad j = 1,\dots,k. \tag{1.33}$$

Other choices can define completely $\{\Psi_j(x)\}_{j=1}^k$, by specifying the other k conditions, for example,

$$\frac{d^m \Psi_j(\rho_r)}{dx^m} = \delta_{j,r} \qquad r = 1,\dots,k,$$

where $\delta_{j,r}$ is the Kronecker delta. This last choice, used in [70], yields a collocation scheme that is equivalent to using an implicit Runge Kutta method [12]. Its benefits over the choice (1.33) have been shown in [9]. To construct the corresponding monomial basis of $\mathbb{S}_\Delta$, we impose m continuity conditions on each internal x_i, for $i = 1,\dots,N-1$. Thus, let $u_\Delta(x) \in \mathbb{S}_\Delta$, then there exists some $\{z_{i,j}\}_{1\le i\le N, 1\le j\le m}$ and $\{w_{i,j}\}_{1\le i\le N, 1\le j\le k}$ such that for any $x \in [x_{i-1}, x_i]$, $i = 1,\dots,N$, we get $u_\Delta(x) = u_{i,\Delta}(x)$ where

$$u_{i,\Delta}(x) := \sum_{j=1}^{m} z_{i,j}\Phi_j(x-x_{i-1}) + h_i^m \sum_{j=1}^{k} w_{i,j}\Psi_j\left(\frac{x-x_{i-1}}{h_i}\right), \tag{1.34}$$

and where $\{z_{i,j}\}_{1\le i\le N,1\le j\le m}$ and $\{w_{i,j}\}_{1\le i\le N,1\le j\le k}$ are chosen such that the $m(N-1)$ continuity conditions

$$\frac{d^{l-1}u_{i,\Delta}}{dx^{l-1}}(x_i) = \frac{d^{l-1}u_{i+1,\Delta}}{dx^{l-1}}(x_i), \qquad i = 1,\dots,N-1 \text{ and } l = 1,\dots,m \tag{1.35}$$

are satisfied. This yields

$$z_{i,r} = D^{r-1}u_\Delta(x_{i-1}), \qquad i = 2,\dots,N \text{ and } r = 1,\dots,m.$$

The remaining

$$(k+m)N - m(N-1) = kN + m$$

free parameters in the representation (1.34) are determined by imposing on $u_\Delta(x)$ the conditions (C3)-(C4). To obtain a linear system with BABD structure, we choose m new variables

$$z_{N+1,r} := D^{r-1}u_\Delta(x_N), \qquad r = 1,\dots,m.$$

Thus, the boundary conditions (C4) are equivalent to

$$B_a\mathbf{z}_1 + B_b\mathbf{z}_{N+1} = \mathbf{d}, \tag{1.36}$$

and the kN collocation conditions (C3) on the points $\xi_{i,j}$ are equivalent to

$$H_i\mathbf{z}_i + G_i\mathbf{w}_i = \mathbf{f}_i,$$

where $H_i = (H_i^{j,r})$, $G_i = (G_i^{j,r})$ with

$$\begin{aligned} H_i^{j,r} &= L(\Phi_r)(h_i\rho_j), \quad j = 1,\dots,k \text{ and } \quad r = 1,\dots,m,\\ G_i^{j,r} &= h_i^m L(\Psi_r)(\rho_j), \quad j = 1,\dots,k \text{ and } \quad r = 1,\dots,k, \end{aligned}$$

and

$$\begin{aligned} \mathbf{f}_i^T &= (f(\xi_{i,1}),\dots,f(\xi_{i,k}))^T,\\ \mathbf{z}_i^T &= (z_{i,1},\dots,z_{i,m})^T,\\ \mathbf{w}_i^T &= (w_{i,1},\dots,w_{i,k})^T. \end{aligned}$$

The $(N-1)m$ continuity equations (1.35) are equivalent to

$$-E_i\mathbf{z}_i - F_i\mathbf{y}_i + \mathbf{z}_{i+1} = 0, \qquad i = 1,\dots,N.$$

For the definition of $\mathbf{z}_{N+1}$ we have

$$-E_N\mathbf{z}_N - F_N\mathbf{y}_N + \mathbf{z}_{N+1} = 0,$$

where $E_i = (E_i^{j,r})$, $F_i = (F_i^{j,r})$ with

$$\begin{aligned} E_i^{j,r} &= D^{j-1}\Phi_r(h_i) = \frac{h_i^{r-j}}{(r-j)!}, \quad j = 1,\dots,m \text{ and } \quad r = 1,\dots,m,\\ F_i^{j,r} &= h_i^{m+1-j}D^{j-1}\Psi_r(1), \qquad\quad j = 1,\dots,m \text{ and } \quad r = 1,\dots,k. \end{aligned}$$

Note that E_i is an upper triangular matrix. The resulting BABD linear system has the structure

$$\begin{pmatrix} B_a & & & & & & B_b \\ H_1 & G_1 & & & & & \\ -E_1 & -F_1 & I & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & H_N & G_N & \\ & & & & -E_N & -F_N & I \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{w}_1 \\ \mathbf{z}_2 \\ \vdots \\ \vdots \\ \mathbf{z}_N \\ \mathbf{w}_N \\ \mathbf{z}_{N+1} \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ 0 \\ \vdots \\ \mathbf{f}_N \\ 0 \end{pmatrix}. \tag{1.37}$$

with the ith row block

$$V_i = \begin{pmatrix} H_i & G_i & \\ -E_i & -F_i & I \end{pmatrix}.$$

Chapter 2

Algorithms for ABD and BABD systems

We discuss the most important numerical techniques for solving ABD systems and BABD systems. We consider ABD and BABD coefficient matrices in the form shown in Figure 2 and 3, respectively, except in paragraph 2.5.1. We measure and compare the computational cost of these techniques in terms of the total number of *flops*, where each flop represents one of the four arithmetic floating point operations.

Most of the algorithms are effective and stable only for the solution of ABD systems, because the effect of the right-upper block in the BABD structures could result in instability, high computational cost and fill-in. For this reason, codes, like *SOLVEBLOK*, *COLROW* and *ABDPACK* are written for solving only ABD linear systems. Since there were no stable serial available algorithms designed specifically for BABD systems, we transform BABD systems into equivalent ABD systems, see Section 2.2, which can be solved stably with a stable existing ABD solver.

Instability for BABD systems can arise for techniques based on LU factorization, as results in the following example.

2.1 The Wright example

The Wright example [77] shows how a simple well conditioned BABD matrix

$$\bar{A} = \begin{pmatrix} I & & & & I \\ -C & I & & & \\ & -C & I & & \\ & & \ddots & \ddots & \\ & & & -C & I \end{pmatrix}, \tag{2.1}$$

may give rise to instability when it is factored by means of the LU factorization

with row partial pivoting. If all the elements of C have modulus less than 1 then no permutation is ever performed in the partial pivoting algorithm and the right-upper corner block implies fill-in in the last block column of the matrix U, with elements $C, C^2, C^3, \ldots$

$$L = \begin{pmatrix} I & & & & \\ -C & I & & & \\ & -C & I & & \\ & & \ddots & \ddots & \\ & & & -C & I \end{pmatrix} \quad U = \begin{pmatrix} I & & & I \\ & I & & C \\ & & \ddots & \vdots \\ & & & I & C^{N-1} \\ & & & & I + C^N \end{pmatrix}. \tag{2.2}$$

The resulting fill-in is of size Nm^2. If at least one of the eigenvalues of C has modulus greater than 1, then there is an exponential growth in the elements of the last block column of U, and the solution becomes incorrect even if a moderate number of meshpoints $(N + 1)$ is used.

As an example, the matrix (2.1) can arise from the numerical solution of the well conditioned BVODE with non-separated BCs

$$\begin{aligned} & y'(x) = Ay(x) + r(x), \quad x \in [0, 60], \\ & y(0) + y(60) = \eta, \\ & A = \begin{pmatrix} -\frac{1}{6} & 1 \\ 1 & -\frac{1}{6} \end{pmatrix}, \end{aligned} \tag{2.3}$$

by means of multiple shooting, discussed in Section 1.4. Hence, C becomes the fundamental solution e^{hA}, see Section 1.1.2, where h is the stepsize of each of N subintervals used to discretize the ODE problem. Also, the matrix $\bar{A}$ has a condition number in the infinity norm $cond_\infty(\bar{A})$ which satisfies

$$cond_\infty(\bar{A}) \leq (\|e^{hA}\| + 1)(1 + N). \tag{2.4}$$

If $h = 0.3$ (and $N = \frac{60}{h} = 200$) then

$$C = e^{hA} = \begin{pmatrix} 0.99436 & 0.28967 \\ 0.28967 & 0.99436 \end{pmatrix},$$

and matrix $\bar{A}$ in (2.1) is well conditioned because the bound in (2.4) is equal to 459. Since the elements of C have modulus less than 1, the LU factorization with row partial pivoting in (2.2) is obtained. We note that A in (2.3) has eigenvalues $\lambda_1 = -\frac{7}{6}$, $\lambda_2 = \frac{5}{6}$ with respectively eigenvectors $\mathbf{v}_1 = \begin{pmatrix} 0.7071 \\ -0.7071 \end{pmatrix}$ and $\mathbf{v}_2 = \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$. Therefore, C has eigenvalues $e^{h\lambda_1} = 0.7047$, and $e^{h\lambda_2} = 1.284$.

Since one of the eigenvalues is greater than 1, the LU factorization becomes an unstable procedure. In fact, we note that the elements of C^N are very large

$$C^N = V \begin{pmatrix} e^{h\lambda_1 N} & \\ & e^{h\lambda_2 N} \end{pmatrix} V^T = V \begin{pmatrix} 4 \cdot 10^{-31} & \\ & 5 \cdot 10^{21} \end{pmatrix} V^T$$
$$= 10^{21} \begin{pmatrix} -2.6 & 2.6 \\ -2.6 & 2.6 \end{pmatrix},$$

where V is the matrix $(\mathbf{v_1}, \mathbf{v_2})$ having the eigenvectors of A as columns.

The same drawback appears when (2.3) is approximated by means of the trapezoidal rule, thus obtaining a BABD coefficient matrix (2.1) with $C = (I - hA/2)^{-1}(I + hA/2)$.

For this reason, Garrett and Gladwell in [38] suggest solving BABD systems by means of an orthogonal factorization, thus obtaining a stable factorization but with a larger computational cost and memory requirement. In fact, QR is much more expensive than LU factorization and, in addition, since this algorithm is not specifically designed for BABD matrices, it implies fill-in on the last block column and on a further upper diagonal of the matrix R of the factorization. In such a case, a block scaling of the coefficient matrix (in order to obtain some blocks equal to the identity) before computing the QR factorization, reduces the number of operations but may be dangerous if performed on ill conditioned blocks.

2.2 Rewriting a BABD system as an ABD system

The techniques, that will be described in this chapter, except for RSCALE presented in Section 2.6, have been designed for solving ABD linear systems directly. However, they can be also used for BABD systems by rewriting them as ABD systems of approximately twice the size. Thus, if we use a stable ABD algorithm, it is possible to solve a BABD linear system through the solution of an equivalent ABD linear system.

This transformation is based on a BVODE technique, cited in [11], that converts nonseparated boundary conditions into separated form. In fact, we start by supposing that the BABD linear system (in Figure 3) arises from the discretization of a nonlinear BVP

$$\begin{aligned} &\mathbf{y}' = \mathbf{f}(x, \mathbf{y}(x)), \qquad x \in [a, b], \\ &D_a \mathbf{y}(a) + D_b \mathbf{y}(b) = \mathbf{d}. \end{aligned} \tag{2.5}$$

We add trivial differential equation with associated initial condition:

$$\mathbf{z}' = \mathbf{0}, \quad \mathbf{z}(a) = \mathbf{y}(a). \tag{2.6}$$

Let $\Delta = \{x_i\}_{i=0}^N$ be the partition (1.15) of $[a, b]$ associated with (2.5), and $x_0 := a$, we get from (2.6)

$$\begin{aligned} &\mathbf{z}(x_i) - \mathbf{z}(x_{i-1}) = 0, \qquad i = 1, \ldots, N-1, \\ &\mathbf{z}(x_0) = \mathbf{y}(x_0). \end{aligned}$$

Then, from $\mathbf{z}_i := \mathbf{z}(x_i)$ the boundary conditions $D_a\mathbf{y}_0 + D_b\mathbf{y}_N = \mathbf{d}$ can be set in the form

$$D_a\mathbf{z}_N + D_b\mathbf{y}_N = \mathbf{d},$$

and the resulting ABD linear system is

$$\begin{pmatrix} -I & I & & & & & & & \\ S_0 & 0 & R_1 & & & & & & \\ & -I & 0 & I & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & & S_{N-1} & 0 & R_N & & \\ & & & & & -I & 0 & I & \\ & & & & & & D_b & D_a & \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{z}_0 \\ \mathbf{y}_1 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{y}_N \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{f}_1 \\ 0 \\ \vdots \\ \mathbf{f}_N \\ 0 \\ \mathbf{d} \end{pmatrix}. \tag{2.7}$$

This block tridiagonal system may be solved using standard, stable linear algebra algorithms either for banded systems or for ABD systems.

2.3 Gaussian elimination with partial pivoting

The algorithm applies Gaussian elimination with row partial pivoting (in the following, referred to as conventional elimination) to an ABD linear system, represented in Figure 2. It is implemented in *SOLVEBLOK* [20, 21]. Similar results can be obtained with column partial pivoting. The package requires $(2N + 1)m^2 + 2Nm_{top}m$ memory allocations $A_1, A_2, \ldots, A_N \in \mathbb{R}^{(m+m_{top})\times 2m}$, and $A_{N+1} \in \mathbb{R}^{m\times m}$

$$A_1 = \begin{pmatrix} \boxed{D_{top}} & * \\ \boxed{V_1} & \end{pmatrix},$$

$$A_i = \begin{pmatrix} * & * \\ \boxed{V_i} & \end{pmatrix}, \quad i = 2, \ldots, N,$$

$$A_{N+1} = \begin{pmatrix} * \\ \boxed{D_{bot}} \end{pmatrix},$$

which store the block rows

$$V_i = \begin{pmatrix} S_{i-1} & R_i \end{pmatrix} \in \mathbb{R}^{m\times 2m}, \qquad i = 1, \ldots, N,$$
$$D_{top} \in \mathbb{R}^{m_{top}\times m}, \quad D_{bot} \in \mathbb{R}^{(m-m_{top})\times m},$$

of the given ABD linear systems, represented in Figure 2. The resulting fill-in (that is, the extra memory requirement) is of size $2Nm_{top}m$. In fact, in each A_i, $i = 2, \ldots, N$, there are $2m_{top}m$ extra allocations, represented by $\begin{pmatrix} * & * \end{pmatrix}$, over

each block V_i, and in A_1, A_{N+1} there are arrays $\begin{pmatrix} * \end{pmatrix}$ of size $m_{top} \times m$, which are filled by the resulting factorization of the algorithm.

In *SOLVEBLOK*, the $N+1$ blocks A_i are stored consecutively in a vector of length $(2N+1)m^2 + 2Nm_{top}m$, containing first all the elements of A_1 followed by those of A_2 and so on.

The elimination phase in *SOLVEBLOK* has the following algorithm:

1. Apply m steps of conventional elimination to A_1;

2. for $i = 2, \ldots, N$

   a) Take the m_{top} rows not yet used in the factorization of A_{i-1} followed by the first $m - m_{top}$ (non-null) rows of A_i and shift them into the first $m \times m$ memory allocations of A_i,

   b) Apply m steps of conventional elimination to A_i,

   end;

3.

   a) Take the m_{top} rows not yet used in the factorization of A_N followed by the first $m - m_{top}$ (non-null) rows of A_{N+1} and shift them into the $m \times m$ memory allocations of A_{N+1},

   b) Apply $m - 1$ steps of conventional elimination to A_{N+1},

The computational cost of the elimination phase is

$$N\left(\frac{5}{3}m^3 + 3m_{top}m^2 - \frac{3}{2}m^2 - \frac{1}{6}m\right) + \left(\frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m\right),$$

and the computational cost of the back-substitution phase is

$$N(4m^2 + 2m_{top}m - m).$$

2.4 Alternating row and column elimination

Based on an approach of Varah [74], alternating row and column elimination, also denoted by ARCE, generates no fill-in for ABD linear systems. We consider an

ABD coefficient matrix with the structure:

$$A = \begin{pmatrix} D_{top} & & & & \\ V_1 & & & & \\ & V_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & V_N & \\ & & & & D_{bot} \end{pmatrix}. \tag{2.8}$$

In contrast of conventional elimination described as in the previous section, we generate no fill-in alternating row eliminations with column eliminations. First, m_{top} column eliminations with column interchanges are performed in the first m_{top} columns of the ABD matrix. Then, we obtain the matrix $A^{(1)} \in \mathbb{R}^{m\times(2m-m_{top})}$ in V_1, and the matrix $\begin{pmatrix} C^{(0)} \\ M^{(1)} \end{pmatrix} \in \mathbb{R}^{(m_{top}+m)\times m_{top}}$ with $C^{(0)} \in \mathbb{R}^{m_{top}\times m_{top}}$ a lower triangular matrix and $M^{(1)} \in \mathbb{R}^{m\times m_{top}}$, see Figure 2.1. As in conventional

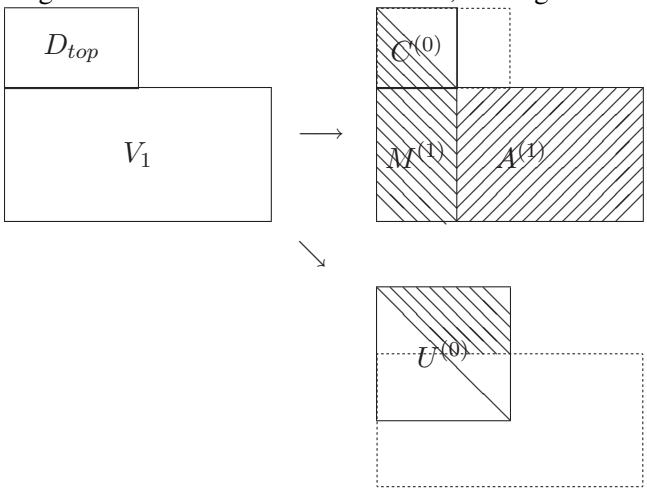

Figure 2.1: First m_{top} column eliminations, the shaded area represents potentially nonzero elements and the dotted lines represents the profile of the original row blocks

column elimination[2] procedure, the multipliers associated are stored in the first m_{top} rows of $U^{(0)} \in \mathbb{R}^{m\times m}$ a unit upper triangular matrix placed in the first m_{top} columns of the ABD matrix, see Figure 2.1. Thus, no fill-in is generated.

[2]We refer to conventional column elimination the Gaussian elimination where column elimination with column interchanges are performed instead of row eliminations with row interchanges as in row partial pivoting.

The algorithm proceeds with $m-m_{top}$ row eliminations with row interchanges, that eliminates rows in $A^{(1)}$ without any fill-in. In fact, the rows of the adjacent block V_2 are not affected by these row interchanges. This is a major benefit with respect to conventional elimination described in Section 2.3.

This step is applied to the row block $\begin{pmatrix} M^{(1)} & A^{(1)} \end{pmatrix}$ and generates: the matrix $A^{(2)} \in \mathbb{R}^{m_{top} \times m}$ and $\begin{pmatrix} R^{(1)} & N^{(1)} \end{pmatrix}$ (corresponding to the $m - m_{top}$ eliminated rows in $A^{(1)}$) with $R^{(1)} \in \mathbb{R}^{(m-m_{top})\times(m-m_{top})}$ an upper triangular matrix and $N^{(1)} \in \mathbb{R}^{(m-m_{top})\times m}$, see Figure 2.2.

Since the row eliminations are applied to $M^{(1)}$, this block is modified and will be denoted by $\tilde{M}^{(1)}$, for future reference. The multipliers associated with the m_{top} row eliminations are stored in the first $m - m_{top}$ columns of the unit lower triangular matrix $L^{(1)} \in \mathbb{R}^{m\times(m-m_{top})}$.

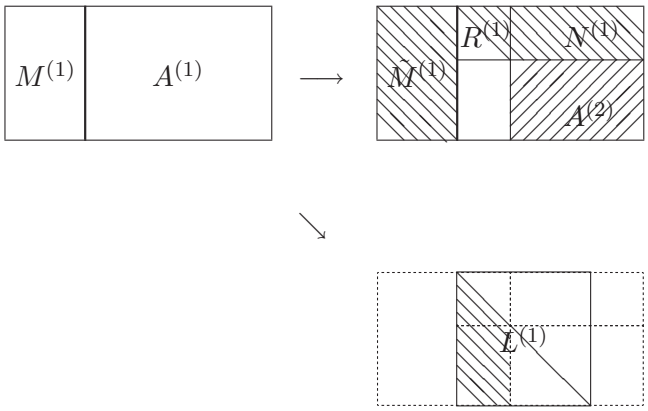

Figure 2.2: $m - m_{top}$ row eliminations, the shaded area represents potentially nonzero elements and the dotted lines represent the profile of the original row block

We repeat this process:

1. m_{top} column eliminations with column interchanges,
2. $m - m_{top}$ row eliminations with row interchanges,

until elimination takes place in the block D_{bot}, at this point $m - m_{top}$ row eliminations with row interchanges are performed. These row eliminations determine the blocks: $\tilde{M}^{(N)}$ of size $(m-m_{top})\times m_{top}$, $R^{(N+1)}$ of size $(m-m_{top})\times(m-m_{top})$ and $L^{(bot)} \in \mathbb{R}^{(m-m_{top})\times(m-m_{top})}$ a unit lower triangular matrix. We note that, the ith column elimination step modifies also the coefficients of the block $N^{(i-1)}$ that will be denoted by $\tilde{N}^{(i-1)}$.

In matrix form, this strategy leads to a decomposition

$$A = PL\tilde{\mathcal{B}}UQ \tag{2.9}$$

where

- P is a permutation matrix recording the row interchanges, with the following structure

$$P = \begin{pmatrix} I & & & & \\ & P^{(1)} & & & \\ & & \ddots & & \\ & & & P^{(N)} & \\ & & & & P^{(bot)} \end{pmatrix},$$

with I is the identity matrix of dimension m_{top}, $P^{(i)} \in \mathbb{R}^{m \times m}$ and $P^{(bot)} \in \mathbb{R}^{(m-m_{top}) \times (m-m_{top})}$.

- L is the unit lower triangular matrix

$$L = \begin{pmatrix} I & & & & \\ & L^{(1)} & & & \\ & & \ddots & & \\ & & & L^{(N)} & \\ & & & & L^{(bot)} \end{pmatrix},$$

containing the row elimination multipliers. Each $L^{(i)}$ is a square unit lower triangular matrix of dimension m, see Figure 2.2, and $L^{(bot)}$ is a square unit lower triangular matrix of dimension $m - m_{top}$. The identity matrix I is of dimension m_{top}.

- $\tilde{\mathcal{B}}$ is the reduced matrix

$$\tilde{\mathcal{B}} = \begin{pmatrix} \tilde{B}_{top} & & & & & \\ \tilde{B}_{1,1} & \tilde{B}_{1,2} & & & & \\ & \tilde{B}_{2,1} & \tilde{B}_{2,2} & & & \\ & & & \ddots & & \\ & & & & \tilde{B}_{N,1} & \tilde{B}_{N,2} \\ & & & & & \tilde{B}_{bot} \end{pmatrix}, \tag{2.10}$$

where

$$\tilde{B}_{i,1} = \begin{pmatrix} \tilde{M}^{(i)} & R^{(i)} \\ & O \end{pmatrix}, \qquad M^{(i)} \in \mathbb{R}^{m \times m_{top}}, R^{(i)} \in \mathbb{R}^{(m-m_{top}) \times (m-m_{top})},$$

$$\tilde{B}_{i,2} = \begin{pmatrix} \tilde{N}^{(i)} \\ C^{(i)} \quad O \end{pmatrix}, \qquad \tilde{N}^{(i)} \in \mathbb{R}^{(m-m_{top}) \times m}, C^{(i)} \in \mathbb{R}^{m_{top} \times m_{top}},$$

and

$$\tilde{B}_{top} = \begin{pmatrix} C^{(0)} & O \end{pmatrix}, \qquad \tilde{B}_{bot} = \begin{pmatrix} \tilde{M}^{(N+1)} & R^{(bot)} \end{pmatrix}.$$

- U is a unit upper triangular matrix

$$U = \begin{pmatrix} U^{(0)} & & & \\ & U^{(1)} & & \\ & & \ddots & \\ & & & U^{(N)} \end{pmatrix},$$

  which contains the column elimination multipliers. Each $U^{(i)}$ is a square unit upper triangular matrix of dimension m, see Figure 2.1.

- Q is a permutation matrix recording the column interchanges, with the following structure

$$Q = \begin{pmatrix} Q^{(0)} & & & \\ & Q^{(1)} & & \\ & & \ddots & \\ & & & Q^{(N)} \end{pmatrix},$$

  with $Q^{(i)} \in \mathbb{R}^{m\times m}$, for $i = 0, \ldots, N$.

The solution of $A\mathbf{y} = \mathbf{f}$ with the ABD coefficient matrix A of dimension $(N+1)m$ decomposed as in (2.9) consists in five steps:

$$\begin{aligned} &1) && P\mathbf{y}^{(1)} = \mathbf{f}, \\ &2) && L\mathbf{y}^{(2)} = \mathbf{y}^{(1)}, \\ &3) && \tilde{\mathcal{B}}\mathbf{y}^{(3)} = \mathbf{y}^{(2)}, \\ &4) && U\mathbf{y}^{(4)} = \mathbf{y}^{(3)}, \\ &5) && Q\mathbf{y} = \mathbf{y}^{(4)}. \end{aligned} \tag{2.11}$$

The system in step 3) is solved using a forward recursion for some unknowns and a backward recursion for the others. In the forward recursion, the lower triangular linear systems in $\tilde{\mathcal{B}}\mathbf{y}^{(3)} = \mathbf{y}^{(2)}$ involving the coefficient matrices $C^{(i)}$ are solved, starting at $i = 0$. In the backward recursion, upper triangular linear systems associated with the coefficient matrices $R^{(i)}$ are solved, starting by the use of $R^{(bot)}$ in the last row of $\tilde{\mathcal{B}}$, see (2.10).

The computational cost of the elimination phase is

$$N(\tfrac{5}{3}m^3 + 4m^2 m_{top} - 4m m_{top}^2 - \tfrac{3}{2}m^2 - m m_{top} + m_{top}^2 - \tfrac{1}{6}m)$$
$$+\tfrac{2}{3}m^3 + m^2 m_{top} - \tfrac{1}{2}m^2 - m m_{top} - \tfrac{1}{6}m.$$

and the computational cost of the solution phase (the five steps 1-5) is

$$N(4m^2 - m) + 2m^2 - m.$$

2.5 Modified alternate row and column elimination

A modification of ARCE has been developed and implemented in the code *COLROW* for ABD linear systems with block rows of the same size and in *ARCECO* for general ABD linear systems with block rows of different size. This modification, called MARCE, reduces the computational cost with respect to ARCE by applying the eliminations only to part of the matrix obtained with ARCE as in the previous section.

For example, considering the first m_{top} column eliminations, the resulting matrix has the form

$$\mathcal{B}_1 = AQ_1U_1 = \left(\begin{array}{c|c} C^{(0)} & O \\ \hline M^{(1)} & \\ & A_1 \\ O & \end{array}\right), \tag{2.12}$$

where the blocks $C^{(0)}$, $M^{(1)}$ are defined in Section 2.4. The matrix A_1 is square of dimension $(N+1)m - m_{top}$ and is almost block diagonal. Q_1 is a permutation matrix (of size $(N+1)m \times (N+1)m$), associated with the column interchanges. The matrix

$$U_1 = \left(\begin{array}{cc} {U^{(0)}}^{-1} & \\ & I \end{array}\right) \in \mathbb{R}^{(N+1)m \times (N+1)m}$$

contains the multipliers associated with the column eliminations. This procedure applies the successive $m - m_{top}$ row eliminations only to the submatrix A_1 and therefore with respect to ARCE, we save the computations used there for modifying $M^{(1)}$. The second step yields

$$L_1P_1A_1 = \left(\begin{array}{c|cc} R^{(1)} & N^{(1)} & O \\ \hline O & \multicolumn{2}{c}{A_2} \end{array}\right),$$

where the blocks $R^{(1)}$, $N^{(1)}$ are defined in Section 2.4. The matrix A_2 is square and almost block diagonal. Q_1 is a permutation matrix associated with the row interchanges, the matrix

$$L_1 = \left(\begin{array}{cc} {L^{(1)}}^{-1} & \\ & I \end{array}\right)$$

contains the multiplier associated with the row eliminations.

Analogously, when in the next step, m_{top} column eliminations are used, they are applied only to A_2 and therefore $N^{(1)}$ is not changed. Finally, after N successive steps, each formed by with m_{top} column eliminations and $m - m_{top}$ row eliminations, the following reduced matrix $\mathcal{B}$ is obtained

$$\mathcal{B} = \begin{pmatrix} B_{top} & & & & & \\ B_{1,1} & B_{1,2} & & & & \\ & B_{2,1} & B_{2,2} & & & \\ & & & \ddots & & \\ & & & & B_{N,1} & B_{N,2} \\ & & & & & B_{bot} \end{pmatrix}, \tag{2.13}$$

where $B_{i,1} = \left(\begin{array}{|c|c|} \hline & R^{(i)} \\ M^{(i)} & \\ & O \\ \hline \end{array} \right)$, $B_{i,2} = \left(\begin{array}{|c|} \hline N^{(i)} \\ \hline C^{(i)} \quad \mathrm{O} \\ \hline \end{array} \right)$, $i = 1, \ldots, N,$

$$B_{top} = \begin{pmatrix} C^{(0)} & O \end{pmatrix}, \qquad B_{bot} = \begin{pmatrix} M^{(N+1)} & R^{(bot)} \end{pmatrix}.$$

This structure corresponds to that of $\tilde{\mathcal{B}}$ in (2.13), with $M^{(i)}$ and $N^{(i)}$ in place of $\tilde{M}^{(i)}$ and $\tilde{N}^{(i)}$, respectively.

The decomposition of the ABD coefficient matrix A, obtained with this procedure, cannot be written in the form (2.9) since eliminations involve only parts of the rows or columns. Therefore, the system

$$A\mathbf{y} = \mathbf{f},$$

with

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} \mathbf{d}_a \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \\ \mathbf{d}_b \end{pmatrix},$$

is solved with a different procedure to that used for ARCE in Section 2.4. The steps 1)-5) in (2.11) are performed with the operations summarized in the algorithm in Figure 2.3.

The computational cost of the elimination phase is:

$$\begin{array}{l} N\left(\frac{5}{3}m^3 + m^2 m_{top} - m m_{top}^2 + -\frac{3}{2}m^2 + m m_{top} - m_{top}^2 - \frac{1}{6}m\right) \\ +\frac{2}{3}m^3 + 2m m_{top}^2 - m_{top}^3 - \frac{1}{2}m^2 - m_{top}^2 - \frac{1}{6}m \end{array}$$

We save with respect to ARCE

$$N(3m m_{top}(m - m_{top}) - 2m_{top}(m - m_{top})) + m_{top}(m - m_{top})(m - m_{top} - 1)$$

% Solution phase

let $\mathbf{y}_i = \begin{pmatrix} \mathbf{y}_{22,i} \\ \mathbf{y}_{3,i} \end{pmatrix}$

 % Forward recursion

  solve $C^{(0)}\mathbf{x}_1^{(0)} = \mathbf{d}_a$

  for $i = 1, \ldots, N$

    compute $\mathbf{c}^{(i)} = \mathbf{f}_i - M^{(i-1)}\mathbf{x}_1^{(i-1)}$

    solve $P^{(i)}L^{(i)}\mathbf{d}^{(i)} = \mathbf{c}^{(i)}$

    solve $C^{(i)}\mathbf{x}_1^{(i)} = \mathbf{d}_2^{(i)}$, where $\mathbf{d}^{(i)} = \begin{pmatrix} \mathbf{d}_1^{(i)} \\ \mathbf{d}_2^{(i)} \end{pmatrix}$

  end

  compute $\mathbf{c}^{(bot)} = \mathbf{d}_b - M^{(N)}\mathbf{x}_1^{(N)}$

  solve $P^{(bot)}L^{(bot)}\mathbf{d}^{(bot)} = \mathbf{c}^{(bot)}$

 % Backward recursion

  solve $R^{(bot)}\mathbf{x}_2^{(N)} = \mathbf{d}^{(bot)}$

  for $i = N, \ldots, 1$

    solve $U^{(i)}Q^{(i)}\mathbf{y}_{3,i} = \mathbf{x}^{(i)}$, where $\mathbf{x}^{(i)} = \begin{pmatrix} \mathbf{x}_1^{(i)} \\ \mathbf{x}_2^{(i)} \end{pmatrix}$

    compute $\mathbf{e}^{(i)} = \mathbf{d}_1^{(i)} - N^{(i)}\mathbf{y}_{3,i}$

    solve $R^{(i)}\mathbf{y}_{2,i} = \mathbf{e}^{(i)}$, where $\mathbf{y}_{2,i} = \begin{pmatrix} \mathbf{x}_2^{(i-1)} \\ \mathbf{y}_{22,i} \end{pmatrix}$

  end

  solve $U^{(0)}Q^{(0)}\mathbf{y}_0 = \begin{pmatrix} \mathbf{x}_1^{(0)} \\ \mathbf{x}_2^{(0)} \end{pmatrix}$

Figure 2.3: Solution phase for the technique MARCE

flops. This number of flops is greater than zero because $m > m_{top}$.

The computational cost of the solution phase, described in the algorithm in Figure 2.3, is the same of that described for ARCE, that is

$$N(4m^2 - m) + 2m^2 - m.$$

2.5.1 The *ABDPACK* algorithm

The techniques described above can be applied to ABD linear systems arising from spline collocation with the structure in (1.32). For this system, *COLROW* does not introduce any fill-in, in contrast *SOLVEBLOK* does. See [29], where the MARCE technique used in *COLROW* and applied to systems with structure (1.32) is described in more detail.

Now, let

$$\begin{pmatrix} D_{top} & & & & & & \\ H_1 & G_1 & & & & & \\ -E_1 & -F_1 & I & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & H_N & G_N & \\ & & & & -E_N & -F_N & I \\ & & & & & & D_{bot} \end{pmatrix}, \tag{2.14}$$

be the coefficient matrix associated with to linear systems arising from spline collocation using a monomial spline basis, see Section 1.7.2. The blocks in (2.14) have the following sizes: $D_{top} \in \mathbb{R}^{m_{top} \times m}$, $D_{bot} \in \mathbb{R}^{(m-m_{top}) \times m}$, $H_i \in \mathbb{R}^{k \times m}$, $G_i \in \mathbb{R}^{k \times k}$, $E_i \in \mathbb{R}^{m \times m}$, $F_i \in \mathbb{R}^{m \times k}$ and I is the unitary matrix of dimension m. The code *SOLVEBLOK* for the structure in (2.14), requires an extra memory of length $(N-1)(m_{top}(k+2m) + km) + m_{top}m$, see [64]. In contrast, *COLROW* needs Nkm memory locations more than the necessary storage for the matrix (2.14). This fill-in corresponds to the null blocks of size $k \times m$ over each identity matrix I in (2.14).

In 1992, the papers [63, 64] introduce a code, *ABDPACK*, which uses the MARCE technique that, exploiting the special pattern of systems with coefficient matrices (2.14), does not generate fill-in.

The technique has the following phases:

1. perform m_{top} column eliminations with column interchanges in D_{top}
2. in each block row $\begin{pmatrix} H_i & G_i & \\ -E_i & -F_i & I, \end{pmatrix}$ perform
   - k column eliminations with column interchanges,
   - $m - m_{top}$ row eliminations with row interchanges,

- row interchanges to preserve the lower triangular form of the block corresponding to the identity matrix in the row block,
- m_{top} column eliminations with column interchanges,

3. $m - m_{top} - 1$ row eliminations on the bottom block D_{bot}.

The computational cost of the elimination phase is:

$$\begin{aligned}&N(\tfrac{2}{3}k^3 + 2k^2m + 2km^2 + \tfrac{5}{3}m^3 - mm_{top}^2 - k^2m_{top} - km_{top}^2 + m^2m_{top} - \tfrac{1}{2}k^2 - km\\ &\quad -\tfrac{3}{2}m^2 - km_{top} + mm_{top} - m_{top}^2 - \tfrac{1}{6}k - \tfrac{1}{6}m)\\ &+2kmm_{top} + \tfrac{2}{3}m^3 - km_{top}^2 + 2mm_{top}^2 - m_{top}^3\\ &-km_{top} - \tfrac{1}{2}m^2 - m_{top}^2 - \tfrac{1}{6}m.\end{aligned}$$

The computational cost for the solution phase is

$$\begin{aligned}&N(m^2 + 2km + 2mm_{top} - k^2 + 2m_{top}^2 + 2m_{top}k - k - m)\\ &+2km - 2m_{top}k - m_{top}^2 - 3m^2 - m\end{aligned}$$

2.6 The *RSCALE* algorithm

The parallel code *PMIRKDC* [68] uses the parallel *RSCALE* algorithm [49] for solving the Jacobian linear systems which are generated by the use of the Mono Implicit Runge Kutta formulae described in Section 1.3. *RSCALE* is based on block eigenvalue rescaling applied to the a BABD linear system with coefficient matrix and right hand side:

$$\left[\begin{array}{ccccc|c} D_a & & & & D_b & \mathbf{d} \\ S_0 & R_1 & & & & \mathbf{f}_1 \\ & S_1 & R_2 & & & \mathbf{f}_2 \\ & & \ddots & \ddots & & \vdots \\ & & & S_{N-1} & R_N & \mathbf{f}_N \end{array}\right], \tag{2.15}$$

where the blocks D_a, D_b, S_i and R_i are of size $m \times m$.

We describe the serial version of the algorithm *RSCALE*, as it is presented in [49]. Numerical tests of *RSCALE* in Section 3.7 are relative to this serial version.

The factorization phase is divided into two levels. In the first level, local transformations are applied to the linear system. These transformations do not modify the dimension of the linear system. When there are row transformations, the right hand side associated with the coefficient matrix is modified consequently.

The first step modifies all the block rows of the coefficient matrix through the matrix multiplication

$$\begin{pmatrix} D_a & & & & D_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} I & -I & & & \\ & I & -I & & \\ & & \ddots & \ddots & \\ & & & I & -I \\ & & & & I \end{pmatrix} =$$

$$\begin{pmatrix} D_a & -D_a & & & & & D_b \\ S_0 & R_1 - S_0 & -R_1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & S_{N-2} & R_{N-1} - S_{N-2} & -R_{N-1} & \\ & & & & S_{N-1} & R_N - S_{N-1} & \end{pmatrix}. \quad (2.16)$$

This step can be summarized by the sequence of operations

```
for i = N,...,0
   column_(i+1)=column_(i+1)-column_(i)
end
```

where $\texttt{column}_{(i)}$ is the ith block-column of the coefficient matrix in (2.15). Also, we indicate by $\texttt{row}_{(i)}$ the ith block-row of the coefficient matrix in (2.15). Let

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_0 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} \in \mathbb{R}^{m(N+1)}$$

be the vector of unknowns associated with the initial BABD linear system (2.15), then the linear system associated with the coefficient matrix (2.16) has

$$\tilde{\mathbf{y}} = \begin{pmatrix} I & I & \dots & I \\ & I & \dots & I \\ & & \ddots & \vdots \\ & & & I \end{pmatrix} \mathbf{y} \quad (2.17)$$

as vector of unknowns.

The next two block-operation steps modify only the last two block rows

$$\left[\begin{array}{ccc|c} S_{N-2} & R_{N-1} - S_{N-2} & -R_{N-1} & \mathbf{f}_{N-1} \\ & S_{N-1} & R_N - S_{N-1} & \mathbf{f}_N \end{array}\right]$$

of the linear system with coefficient matrix (2.16):

1. $\texttt{row}_{(N+1)} = (R_N - S_{N-1})^{-1} \ \texttt{row}_{(N+1)}$,
   this operation transforms the block $R_N - S_{N-1}$ (of $\texttt{row}_{(N+1)}$) into the identity matrix I

$$\left[\begin{array}{ccc|c} S_{N-2} & R_{N-1} - S_{N-2} & -R_{N-1} & \mathbf{f}_{N-1} \\ & \tilde{S}_{N-1} & I & \tilde{\mathbf{f}}_N \end{array}\right]$$

   where $\tilde{S}_{N-1} = (R_N - S_{N-1})^{-1} S_{N-1}, \qquad \tilde{\mathbf{f}}_N = (R_N - S_{N-1})^{-1} \mathbf{f}_N.$

2. $\texttt{row}_{(N)} = \texttt{row}_{(N)} + R_{N-1}\ \texttt{row}_{(N+1)}$,
   this operation annihilates $-R_{N-1}$ in $\texttt{row}_{(N)}$

$$\left[\begin{array}{ccc|c} & S_{N-2} & R_{N-1}(I+\tilde{S}_{N-1})-S_{N-2} & \tilde{\mathbf{f}}_{N-1} \\ & & \tilde{S}_{N-1} \quad I & \tilde{\mathbf{f}_N} \end{array}\right]$$

   where $\tilde{\mathbf{f}}_{N-1} = \mathbf{f}_{N-1} + R_{N-1}\tilde{\mathbf{f}_N}$.

The algorithm proceeds by performing the operations of the two steps described above, for each pair of block rows $(\texttt{row}_{(i)}, \texttt{row}_{(i+1)})$ moving upward with i running from $N-1$ to 1.

These operations can be summarized by the sequence

for $i = N-1, \ldots, 0$
  let the pair of block rows $(\texttt{row}_{(i)}, \texttt{row}_{(i+1)})$ be in the form

$$\left[\begin{array}{ccc|c} & S_{i-2} & R_{i-1}-S_{i-2} \quad -R_{i-1} & \mathbf{f}_{i-1} \\ & & S_{i-1} \quad \tilde{R}_i & \tilde{\mathbf{f}}_i \end{array}\right]$$

  scale $\texttt{row}_{(i+1)}$ with respect to $\tilde{R}_i$ through the operation
  $\texttt{row}_{(i+1)} = (\tilde{R}_i)^{-1}\ \texttt{row}_{(i+1)}$

  annihilate $-R_{i-1}$ through the operation
  $\texttt{row}_{(i)} = \texttt{row}_{(i)} + R_{i-1}\texttt{row}_{(i+1)}$
end

The final row operation in this first level is

$$\texttt{row}_{(1)} = \texttt{row}_{(1)} + D_a\texttt{row}_{(2)},$$

which annihilates $-D_a$ in the first row.

The resulting linear system has now the form

$$\left[\begin{array}{ccccc|c} \tilde{D}_a & & & & D_b & \tilde{\mathbf{d}} \\ \tilde{S}_0 & I & & & & \tilde{\mathbf{f}}_1 \\ & \tilde{S}_1 & I & & & \tilde{\mathbf{f}}_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \tilde{S}_{N-1} & I & \tilde{\mathbf{f}_N} \end{array}\right]. \tag{2.18}$$

As mentioned in [49], numerical experiments and preliminary analysis indicate that the blocks $\tilde{S}_i$ in (2.18) have bounded eigenvalues and norms satisfying

$$\begin{array}{ll} 1. & |\lambda_i| \leq 1+\epsilon_1, \qquad \forall \lambda_i \text{ eigenvalue of } \tilde{S}_i, \\ 2. & \|\tilde{S}_i\| \in (1-\epsilon_2, 1+\epsilon_3), \end{array} \tag{2.19}$$

where $i = 1, \ldots, N$, and ϵ_1, ϵ_2, ϵ_3 are positive constants which can be reduced arbitrarily using a suitable choice of parameter during the rescaling phase. In fact,

in the first level of factorization (2.16), the user can choose a suitable parameter σ such that the matrix to multiply to the input BABD coefficient matrix is

$$\begin{pmatrix} I & -\sigma I & & & \\ & I & -\sigma I & & \\ & & \ddots & \ddots & \\ & & & I & -\sigma I \\ & & & & I \end{pmatrix},$$

instead of that resulting from the case $\sigma = 1$ used above.

From properties 1-2 of $\tilde{S}_i$, see (2.19), the system (2.18) can be solved stably using the successive cyclic reduction steps, assumed N is a power of 2:

1. each $\mathtt{row}_{(i)}$ of (2.18), with i even, is premultiplied by $\tilde{S}_{i+1}$ and then it is subtracted from the $\mathtt{row}_{(i+1)}$, obtaining a reduced linear system with a coefficient matrix of size $(\frac{N}{2}+1)m \times (\frac{N}{2}+1)m$

$$\begin{pmatrix} \tilde{D}_a & & & & B_b \\ -\tilde{S}_1\tilde{S}_0 & I & & & \\ & -\tilde{S}_3\tilde{S}_2 & I & & \\ & & \ddots & \ddots & \\ & & & -\tilde{S}_{N-1}\tilde{S}_{N-2} & I \end{pmatrix}, \tag{2.20}$$

2. step (1) is applied to the linear system with coefficient matrix (2.20) obtaining a coefficient matrix of size $(\frac{N}{4}+1)m \times (\frac{N}{4}+1)m$; this step is repeated cyclically $\log_2(N)$ times. This yields a $2m \times 2m$ linear system with coefficient matrix

$$\begin{pmatrix} \tilde{D}_a & D_b \\ (-1)^N \prod_{i=1}^{N} \tilde{S}_{N-i} & I \end{pmatrix}. \tag{2.21}$$

In the solution phase, we compute the solutions of the $\log_2(N)$ linear systems starting with the $2m \times 2m$ linear system with coefficient matrix (2.21), thus, finally determining the solution $\tilde{\mathbf{y}}$, see (2.17), of the linear system (2.18). For more details about back-substitution phase, see Chapter 3.

Finally, the solution $\mathbf{y}$ of the initial linear system (2.15) is determined through the matrix vector product

$$\mathbf{y} = \begin{pmatrix} I & -I & & & \\ & I & -I & & \\ & & \ddots & \ddots & \\ & & & I & -I \\ & & & & I \end{pmatrix} \tilde{\mathbf{y}}. \tag{2.22}$$

The cost of the first level of factorization phase, not involving the operation on the right hand side is $N\left(\frac{14}{3}m^3\right) + 2m^3$, for the second level (the cyclic reduction) is $(N-1)(2m^3 - m^2)$, thus the total cost of the factorization phase is

$$N\left(\frac{20}{3}m^3 - m^2\right) + m^2.$$

The cost for the transformation of the right hand side during the first level of the factorization phase is $N\left(4m^2 - m\right) + 2m^2$. The cost of the reduction of the right hand side during the second level of the factorization is $(N-1)(2m^2 - m)$ and of the multiplication (2.22) is Nm, thus the total cost of the solution phase is

$$N(6m^2 - m) + m.$$

Chapter 3

An algorithm for the solution of BABD systems based on cyclic reduction

We describe the use of the new sequential computing package *BABDCR* [3], [4] for the solution of BABD systems $A\mathbf{y} = \mathbf{f}$ with the following structure

$$\begin{pmatrix} D_a & & & & D_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}. \tag{3.1}$$

This package is based on the cyclic reduction algorithm, originally proposed in [2] to be implemented on a parallel computer. LU factorization is used to reduce the system. It is well known that cyclic reduction applied to block tridiagonal (or ABD) systems is extremely competitive only on parallel computers. However, since the presence of the right-upper block D_b implies fill-in and a higher computational cost in all direct methods, we show that the cyclic reduction algorithm can also be effective on a sequential computer. A parallel version of the package *BABDCR* has also been implemented, see Chapter 5.

Note that, as we have shown in Chapter 1, the BABD structure (3.1), with blocks S_i, R_i, D_a and D_b of the same size, arises frequently in BVP solvers.

3.1 The cyclic reduction algorithm

The idea of this code is to reduce (3.1) cyclically in order to derive systems of lower dimension (involving less unknowns) with the same BABD structure. We suppose that each block in the BABD matrix is of size $m \times m$. Since operations involving the boundary block D_b would create fill-in, all the reduced systems always keep $\mathbf{y}_0$

and $\mathbf{y}_N$ among the unknowns, and leave unchanged the first row of (3.1). For this reason, the first reduction produces the linear system of dimension $\lceil N/2 \rceil + 1$:

$$\begin{pmatrix} D_a & & & & D_b \\ S'_0 & R'_2 & & & \\ & S'_2 & R'_4 & & \\ & & \ddots & \ddots & \\ & & & S'_{N-2} & R'_N \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_2 \\ \mathbf{y}_4 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} \begin{pmatrix} \mathbf{d} \\ \mathbf{f}'_2 \\ \mathbf{f}'_4 \\ \vdots \\ \mathbf{f}'_N \end{pmatrix}.$$

There are several ways to derive the above reduced system from (3.1). But, since the original matrix is essentially block bidiagonal (the first block row is not modified), each row of the reduced block system may be obtained from two adjacent rows in (3.1). That is, for $i = 2j-1$ odd, $j = 1, 2, \ldots, \lfloor N/2 \rfloor$, the subsystem

$$\begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} \begin{pmatrix} \mathbf{y}_{i-1} \\ \mathbf{y}_i \\ \mathbf{y}_{i+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix} \tag{3.2}$$

is transformed in order to obtain one row in the reduced system involving only the unknowns $\mathbf{y}_{i-1}$ and $\mathbf{y}_{i+1}$

$$S'_{i-1}\mathbf{y}_{i-1} + R'_{i+1}\mathbf{y}_{i+1} = \mathbf{f}'_{i+1}. \tag{3.3}$$

If R_i is nonsingular, this can be achieved by multiplying (3.2) on the left by

$$\begin{pmatrix} I & \\ -S_i R_i^{-1} & I \end{pmatrix}, \tag{3.4}$$

thus obtaining (3.3) with $S'_{i-1} = -S_i R_i^{-1} S_{i-1}$, $R'_{i+1} = R_{i+1}$ and $\mathbf{f}'_{i+1} = \mathbf{f}_{i+1} - S_i R_i^{-1}\mathbf{f}_i$.

However, this procedure may be unstable (see section 3.2). For this reason, we perform a partial pivoting LU factorization of the $2m \times m$ matrix containing the overlapping columns:

$$P_i \begin{pmatrix} R_i \\ S_i \end{pmatrix} = \begin{pmatrix} L_i \\ H_i \end{pmatrix} U_i = \begin{pmatrix} I \\ G_i \end{pmatrix} L_i U_i = \begin{pmatrix} I & \\ G_i & I \end{pmatrix} \begin{pmatrix} L_i U_i \\ O \end{pmatrix}, \tag{3.5}$$

where $G_i = H_i L_i^{-1}$ and P_i is a $2m \times 2m$ permutation matrix representing the m row interchanges. The reduction step leads to

$$\begin{aligned} S'_{i-1} &= \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} S_{i-1} \\ O \end{pmatrix}, \\ R'_{i+1} &= \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} O \\ R_{i+1} \end{pmatrix}, \\ \mathbf{f}'_{i+1} &= \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix}. \end{aligned} \tag{3.6}$$

Formulae (3.6) are obtained by multiplying (3.2) by P_i and by the inverse of the lower triangular matrix in (3.5)

$$\begin{pmatrix} I & \\ G_i & I \end{pmatrix}^{-1} = \begin{pmatrix} I & \\ -G_i & I \end{pmatrix}.$$

In fact,

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} = \begin{pmatrix} \widehat{S}_{i-1} & L_iU_i & \widehat{R}_{i+1} \\ S'_{i-1} & & R'_{i+1} \end{pmatrix} \tag{3.7}$$

and

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix} = \begin{pmatrix} \widehat{\mathbf{f}}_i \\ \mathbf{f}'_{i+1} \end{pmatrix}. \tag{3.8}$$

Taking into account the second row in (3.7) and (3.8) yields equation (3.3) of the reduced system. The first row of (3.7) and (3.8)

$$\widehat{S}_{i-1}\mathbf{y}_{i-1} + L_iU_i\mathbf{y}_i + \widehat{R}_{i+1}\mathbf{y}_{i+1} = \widehat{\mathbf{f}}_i$$

is used in the back-substitution phase to compute $\mathbf{y}_i$ from $\mathbf{y}_{i-1}$ and $\mathbf{y}_{i+1}$. The elements $\widehat{S}_{i-1}$, $\widehat{R}_{i+1}$ and $\widehat{f}_i$ are obtained by simple permutations with the matrix P_i. Therefore the total number of nonzero rows in the two blocks $\widehat{S}_{i-1}$ and $\widehat{R}_{i+1}$ is at most m. We note also that in the computation of S'_{i-1} and R'_{i+1} some operations are not actually performed because of the sparsity structure of the matrices involved (i.e. there are some null rows in the matrices multiplied by G_i).

In general, after k steps of reduction, the reduced block matrix has $\lceil N/s \rceil + 1$ block rows, where $s = 2^k$, and it can be further reduced by combining two consecutive rows, for each $i = (2j-1)s$, with $j = 1, 2, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor$

$$\begin{pmatrix} S^{(k)}_{i-s} & R^{(k)}_i & \\ & S^{(k)}_i & R^{(k)}_{i+s} \end{pmatrix} \begin{pmatrix} \mathbf{y}_{i-s} \\ \mathbf{y}_i \\ \mathbf{y}_{i+s} \end{pmatrix} = \begin{pmatrix} \mathbf{f}^{(k)}_i \\ \mathbf{f}^{(k)}_{i+s} \end{pmatrix} \tag{3.9}$$

so as to obtain

$$S^{(k+1)}_{i-s}\mathbf{y}_{i-s} + R^{(k+1)}_{i+s}\mathbf{y}_{i+s} = \mathbf{f}^{(k+1)}_{i+s}. \tag{3.10}$$

In analogy with (3.7) and (3.8), from (3.9) we also deduce $G^{(k)}_i = H^{(k)}_i (L^{(k)}_i)^{-1}$ (which is used in the reduction of the right hand side) and the following equality

$$\widehat{S}^{(k)}_{i-s}\mathbf{y}_{i-s} + L_iU_i\mathbf{y}_i + \widehat{R}^{(k)}_{i+s}\mathbf{y}_{i+s} = \widehat{\mathbf{f}}^{(k)}_i, \tag{3.11}$$

which is used in the back-substitution phase to compute $\mathbf{y}_i$ from $\mathbf{y}_{i-s}$ and $\mathbf{y}_{i+s}$.

The reduction ends after $p = \lceil \log_2(N) \rceil$ steps when the 2×2 block linear system

$$\begin{pmatrix} D_a & D_b \\ S_0^{(p)} & R_N^{(p)} \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_N^{(p)} \end{pmatrix} \tag{3.12}$$

is obtained. The algorithm proceeds with the solution of (3.12) using Gaussian Elimination, obtaining $\mathbf{y}_0$ and $\mathbf{y}_N$. The other unknowns are computed through the back-substitution phase.

The following algorithm summarizes the operations performed:

```
% Factorization of the coefficient matrix
 % Reduction of the coefficient matrix
```

$p = \lceil \log_2(N) \rceil$

```
  for
```

$k = 0, \ldots, p-1$

$s = 2^k$

```
    for
```

$j = 1, 2, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor$

$i = (2j-1)s$

$$P_i \begin{pmatrix} R_i^{(k)} \\ S_i^{(k)} \end{pmatrix} = \begin{pmatrix} L_i \\ H_i \end{pmatrix} U_i$$

$$G_i^{(k)} = H_i L_i^{-1}$$

$$\begin{pmatrix} \widehat{S}_{i-s}^{(k)} \\ S_{i-s}^{(k+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_i^{(k)} & I \end{pmatrix} P_i \begin{pmatrix} S_{i-s}^{(k)} \\ O \end{pmatrix}$$

$$\begin{pmatrix} \widehat{R}_{i+s}^{(k)} \\ R_{i+s}^{(k+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_i^{(k)} & I \end{pmatrix} P_i \begin{pmatrix} O \\ R_{i+s}^{(k)} \end{pmatrix}$$

```
    end
  end
 % Factorization of the final 2 × 2 block linear system
```

$$\widehat{P} \begin{pmatrix} D_a & D_b \\ S_0^{(p)} & R_N^{(p)} \end{pmatrix} = \widehat{L}\widehat{U}$$

```
% Solution of the linear system
 % Reduction of the right hand side
  for
```

$k = 0, \ldots, p-1$

$s = 2^k$

```
    for
```

$j = 1, 2, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor$

$i = (2j-1)s$

$$\begin{pmatrix} \widehat{\mathbf{f}}_i^{(k)} \\ \mathbf{f}_{i+s}^{(k+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_i^{(k)} & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_i^{(k)} \\ \mathbf{f}_{i+s}^{(k)} \end{pmatrix}$$

```
    end
  end
 % Solution of the final 2 × 2 block linear system
```

$$\begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_N \end{pmatrix} = (\widehat{L}\widehat{U})^{-1} \widehat{P} \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_N^{(p)} \end{pmatrix}$$

```
% Back substitution
 for  k = p − 1, . . . , 0
    s = 2^k
    for  j = ⌊⌈N/s⌉/2⌋, . . . , 2, 1
       i = (2j − 1)s
       y_i = (L_i U_i)^{-1} ( f̂_i^(k) − Ŝ_{i−s}^(k) y_{i−s} − R̂_{i+s}^(k) y_{i+s} )
    end
 end
```

The code *BABDCR* requires the storage of: N factorizations [1] $L_i \backslash U_i$ associated with each of the $N-1$ reductions and the factorization of the final 2×2 block linear system (3.12); the m nonzero rows of blocks $\widehat{S}_{i-s}^{(k)}$, $\widehat{R}_{i+s}^{(k)}$, denoted by $F_i^{(k)}$; the $m \times m$ matrices $G_i^{(k)}$. There are $N-1$ matrices $F_i^{(k)}$, $G_i^{(k)}$, each associated with a reduction step.

3.2 Computational cost and required memory

We measure the computational cost of the *BABDCR* algorithm in terms of total number of *flops*, where each flop represents one of the four arithmetic floating point operations.

The factorization of the coefficient matrix requires $N-1$ reductions from (3.9) to (3.10) and the partial pivoting LU factorization of the 2×2 block matrix in (3.12).

The cost of each reduction is $\frac{14}{3}m^3 - \frac{3}{2}m^2 - \frac{1}{6}m$ floating point operations and the 2×2 block factorization $\frac{16}{3}m^3 - 2m^2 - \frac{1}{3}m$ flops. Thus, the factorization requires $\frac{14}{3}m^3 N$ flops to leading order in powers of m and N.

Solving the linear system (3.1) when the matrix is already factorized requires $N-1$ reductions of the right hand side from (3.9) to (3.10), $N-1$ back-substitutions (3.11) to compute the unknown $\mathbf{y}_i$ from $\mathbf{y}_{i-s}$ and $\mathbf{y}_{i+s}$ (previously determined), and the solution of the 2×2 block linear system (3.12) previously factorized. The cost of each reduction is $2m^2$ flops, that of each back–substitution step is $4m^2$, and of the solution of 2×2 block linear system is $8m^2$ flops. Therefore the solution cost of a linear system with the *BABDCR* algorithm is approximately $6m^2 N$ flops.

The computational cost of the *BABDCR* algorithm does not change if the factorization and solution steps are performed together. However, the memory requirement changes dramatically. If the reduction phase of the coefficient matrix and of the right hand side are carried on at the same time, then the relevant part of the factorization of the matrix is stored in place of the original coefficient matrix and the algorithm does not require fill-in. To compute the factorization only once and solve several linear systems, we need to store the permutation matrices,

[1] $L_i \backslash U_i$ denotes a square matrix containing the non null part of the lower triangular matrix L_i and of the upper triangular U_i without its unit diagonal.

all the computed N factorizations $L_i \backslash U_i$ and the $N-1$ blocks $G_i^{(k)}$ and $F_i^{(k)}$. This means that the algorithm requires a fill-in array, that is an extra memory array, of size $m^2(N-1)$ (which is almost one half of the memory required to store the coefficient matrix) and an integer array of size $2mN$. This is because all the factorizations $L_i \backslash U_i$ and $G_i^{(k)}$ can be saved in place of the initial non-factorized matrix that requires $2m^2(N+1)$ allocations.

3.3 Stability

We now discuss the stability of our algorithm in the special case $S_i = S$ and $R_i = R$ for each $i = 1, \ldots, N$. Such a situation arises, for example, when an autonomous BVODE is discretized using constant stepsize. Let λ and x be one eigenvalue and the corresponding eigenvector of the matrix pencil (S, R), see [43]. Then, from (3.5) and (3.6) (the indices i have been neglected because of the constant blocks) we has

$$\begin{aligned} S'x &= \begin{pmatrix} -G & I \end{pmatrix} P \begin{pmatrix} S \\ 0 \end{pmatrix} x = \lambda \begin{pmatrix} -G & I \end{pmatrix} P \begin{pmatrix} R \\ 0 \end{pmatrix} x \\ &= -\lambda \begin{pmatrix} -G & I \end{pmatrix} P \begin{pmatrix} 0 \\ S \end{pmatrix} x = -\lambda^2 \begin{pmatrix} -G & I \end{pmatrix} P \begin{pmatrix} 0 \\ R \end{pmatrix} x \\ &= -\lambda^2 R'x, \end{aligned}$$

where we have used

$$\begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{pmatrix} R \\ S \end{pmatrix} - \begin{pmatrix} 0 \\ S \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} -G & I \end{pmatrix} \mathrm{P} \begin{pmatrix} R \\ S \end{pmatrix} = 0.$$

Hence, as the reduction process goes on, the eigenvalues with modulus less than 1 approach 0 and the eigenvalues with modulus greater than 1 approach infinity. Hence the reduction by means of (3.4) may be unstable because there is no control on the growth (in modulus) of the elements in $S^{(k)}$ (obtained after k steps of reduction). In contrast, the *BABDCR* algorithm should be stable because all the elements of both H_i and L_i have modulus less or equal to 1. Therefore, we expect that the rows of $G_i^{(k)}$ tend quickly to zero, and the matrices $S^{(k)}$ and $R^{(k)}$ to converge to S^* and R^*. Observe that if one row in S^* is nonzero than the corresponding in R^* is zero.

As an example, we analyze the reduction process in the *BABDCR* algorithm on the Wright example [77], see Section 2.1 where the Wright example is presented. The *BABDCR* algorithm does not exhibit instability when it is applied to the Wright problem. In fact, the first step of reduction gives a matrix with the same structure as (2.1) and C^2 instead of C ($P_i = I$ and $G_i = -C$ in (3.5)). Similarly, the successive steps give matrices $S^{(k)}$ with increasing powers of C (C^s, $s = 2^k$, after k steps), until the elements of the blocks become larger than 1 in modulus. At this

point, since row pivoting is performed on the $2m \times m$ block $\begin{pmatrix} I \\ C^s \end{pmatrix}$, the partial pivoting allows control of the growth of the elements of the new reduced matrices. For example, if $h = 0.3$, there is no permutation for the first step and the matrices $S^{(k)}$ and $R^{(k)}$ converge to

$$S^* = \begin{pmatrix} -1.6487 & -1.6487 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad R^* = \begin{pmatrix} 0 & 0 \\ -1 & 1 \end{pmatrix}.$$

Due to the use of the partial pivoting LU factorization, "pathological" cases may occur; however in such cases the growth of the elements of $S^{(k)}$ and $R^{(k)}$ depends on $m \ll N$ and is restricted only to one step of reduction.

3.4 Solution of transposed BABD systems

To determine the condition number [46] of the given BABD coefficient matrix, many packages [54] for linear systems $A\mathbf{y} = \mathbf{f}$ use the factorization of A for solving the transposed system

$$A^T \mathbf{z} = \mathbf{f}, \tag{3.13}$$

see also [50], [51], [52]. We analyze the solution of (3.13) starting from the cyclic reduction described in Section 3.1. The linear system in (3.13) has the structure

$$\begin{pmatrix} D_a^T & S_0^T & & & & \\ & R_1^T & S_1^T & & & \\ & & R_2^T & S_2^T & & \\ & & & \ddots & \ddots & \\ & & & & R_{N-1}^T & S_{N-1}^T \\ D_b^T & & & & & R_N^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}, \tag{3.14}$$

that is, it is still a BABD linear system with boundary blocks in the last row. We prove that cyclic reduction applied to (3.14) gives reduced linear systems with coefficient matrices equal to the transpose of the coefficient matrices obtained by cyclic reduction applied to the original system (3.1). Let $i = 2j - 1$, for $j = 1, 2, \ldots, \lfloor N/2 \rfloor$, from the permutation in (3.5) we perform a partial column pivoting LU factorization of the $(i+1)$th row of the coefficient matrix in (3.14):

$$\begin{aligned} \begin{pmatrix} R_i^T & S_i^T \end{pmatrix} P_i^T &= U_i^T \begin{pmatrix} L_i^T & H_i^T \\ & I \end{pmatrix} \\ &= \begin{pmatrix} U_i^T L_i^T & O \end{pmatrix} \begin{pmatrix} I & G_i^T \\ & I \end{pmatrix}, \end{aligned}$$

where $G_i = H_i L_i^{-1}$. Now, considering the transpose of the matrices in equation (3.7) we obtain

$$\begin{pmatrix} S_{i-1}^T & \\ R_i^T & S_i^T \\ & R_{i+1}^T \end{pmatrix} P_i^T \begin{pmatrix} I & -G_i^T \\ & I \end{pmatrix} = \begin{pmatrix} \widehat{S}_{i-1}^T & {S'_{i-1}}^T \\ U_i^T L_i^T & \\ \widehat{R}_{i+1}^T & {R'_{i+1}}^T \end{pmatrix}, \tag{3.15}$$

where L_i, U_i, S'_{i-1}, R'_{i+1}, $\widehat{S}_{i-1}$, $\widehat{R}_{i+1}$ are the same blocks as in (3.7), and therefore ${S'_{i-1}}^T$ and ${R'_{i+1}}^T$ are the transposes of the blocks (3.6) obtained in the reduction described in Section 3.1. Thus, multiplying (3.15) by $\begin{pmatrix} I & G_i^T \\ & I \end{pmatrix} = \begin{pmatrix} I & -G_i^T \\ & I \end{pmatrix}^{-1}$ and P_i the following equality

$$\begin{pmatrix} S_{i-1}^T & \\ R_i^T & S_i^T \\ & R_{i+1}^T \end{pmatrix} = \begin{pmatrix} \widehat{S}_{i-1}^T & {S'_{i-1}}^T \\ U_i^T L_i^T & \\ \widehat{R}_{i+1}^T & {R'_{i+1}}^T \end{pmatrix} \begin{pmatrix} I & G_i^T \\ & I \end{pmatrix} P_i \tag{3.16}$$

is satisfied. Let

$$\begin{pmatrix} \mathbf{z}'_i \\ \mathbf{z}'_{i+1} \end{pmatrix} = \begin{pmatrix} I & G_i^T \\ & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{z}_i \\ \mathbf{z}_{i+1} \end{pmatrix}, \tag{3.17}$$

where $i = 2j-1$, for $j = 1, 2, \ldots, \lfloor N/2 \rfloor$, using (3.16)-(3.17) the first three rows

$$\begin{pmatrix} D_a^T & S_0^T & & \\ & R_1^T & S_1^T & \\ & & R_2^T & S_2^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}$$

of the system (3.14) are transformed into the form

$$\begin{pmatrix} D_a^T & \widehat{S}_0^T & {S'_0}^T & \\ & U_1^T L_1^T & & \\ & \widehat{R}_2^T & {R'_2}^T & S_2^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}'_1 \\ \mathbf{z}'_2 \\ \mathbf{z}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}. \tag{3.18}$$

Now, we transform sequentially each adjacent triple of rows, for $i = 2j-1$, with $j = 2, \ldots, \lfloor N/2 \rfloor - 1$,

$$\begin{pmatrix} \widehat{R}_{i-1}^T & {R'_{i-1}}^T & {S_{i-1}}^T & & \\ & & R_i^T & S_i^T & \\ & & & R_{i+1}^T & S_{i+1}^T \end{pmatrix} \begin{pmatrix} \mathbf{z}'_{i-2} \\ \mathbf{z}'_{i-1} \\ \mathbf{z}_i \\ \mathbf{z}_{i+1} \\ \mathbf{z}_{i+2} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{i-1} \\ \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix}$$

in the form

$$\begin{pmatrix} \widehat{R}_{i-1}^T & {R'_{i-1}}^T & \widehat{S}_{i-1}^T & {S'_{i-1}}^T & \\ & & U_i^T L_i^T & & \\ & & \widehat{R}_{i+1}^T & {R'_{i+1}}^T & S_{i+1}^T \end{pmatrix} \begin{pmatrix} \mathbf{z}'_{i-2} \\ \mathbf{z}'_{i-1} \\ \mathbf{z}'_i \\ \mathbf{z}'_{i+1} \\ \mathbf{z}_{i+2} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{i-1} \\ \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix}. \tag{3.19}$$

Finally, supposing that N is even, the last term

$$\begin{pmatrix} & \widehat{R}_{N-2}^T & {R'_{N-2}}^T & {S_{N-2}}^T & \\ & & & R_{N-1}^T & S_{N-1}^T \\ D_b^T & & & & R_N^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}'_{N-3} \\ \mathbf{z}'_{N-2} \\ \mathbf{z}_{N-1} \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{N-2} \\ \mathbf{f}_{N-1} \\ \mathbf{f}_N \end{pmatrix}$$

of the system (3.14) yields

$$\begin{pmatrix} & \widehat{R}_{N-2}^T & {R'_{N-2}}^T & \widehat{S}_{N-2}^T & {S'_{N-2}}^T \\ & & & U_{N-1}^T L_{N-1}^T & \\ D_b^T & & & \widehat{R}_N^T & {R'_N}^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}'_{N-3} \\ \mathbf{z}'_{N-2} \\ \mathbf{z}'_{N-1} \\ \mathbf{z}'_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{N-2} \\ \mathbf{f}_{N-1} \\ \mathbf{f}_N \end{pmatrix}. \tag{3.20}$$

The second row of (3.18)-(3.20) is used to compute

$$\mathbf{z}'_i = (L_i U_i)^{-T} \mathbf{f}_i \tag{3.21}$$

and thus to reduce the linear system (3.14) to

$$\begin{pmatrix} D_a^T & {S'_0}^T & & & & \\ & {R'_2}^T & {S'_2}^T & & & \\ & & {R'_4}^T & {S'_4}^T & & \\ & & & \ddots & \ddots & \\ & & & & {R'_{N-2}}^T & {S'_{N-2}}^T \\ D_b^T & & & & & {R'_N}^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}'_2 \\ \mathbf{z}'_4 \\ \vdots \\ \mathbf{z}'_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}'_0 \\ \mathbf{f}'_2 \\ \vdots \\ \mathbf{f}'_N \end{pmatrix}, \tag{3.22}$$

where

$$\begin{aligned} \mathbf{f}'_0 &= \mathbf{f}_0 - \widehat{S}_0^T (L_1 U_1)^{-T} \mathbf{f}_1, \\ \mathbf{f}'_N &= \mathbf{f}_N - \widehat{R}_N^T (L_{N-1} U_{N-1})^{-T} \mathbf{f}_{N-1}, \\ \mathbf{f}'_i &= \mathbf{f}_i - \widehat{R}_i^T (L_{i-1} U_{i-1})^{-T} \mathbf{f}_{i-1} - \widehat{S}_i^T ((L_{i+1} U_{i+1})^T)^{-1} \mathbf{f}_{i+1}, \end{aligned} \tag{3.23}$$

for $i = 2, 4, \ldots, N-2$. The algorithm proceeds with the reduction of the linear system (3.22) using the procedure described for (3.14). The reduction ends after $p = \lceil \log_2(N) \rceil$ steps when the 2×2 block linear system

$$\begin{pmatrix} D_a^T & {S_0^{(p)}}^T \\ D_b^T & {R_N^{(p)}}^T \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_N^{(p)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 \\ \mathbf{f}_N^{(p)} \end{pmatrix} \tag{3.24}$$

is obtained. Observe that the coefficient matrix in (3.24) is the transpose of the 2×2 block coefficient matrix in (3.12) obtained in cyclic reduction applied to $A\mathbf{y} = \mathbf{f}$.

The algorithm proceeds with the solution of (3.24) using Gaussian Elimination, obtaining $\mathbf{z}_0$ and $\mathbf{z}_N^{(p)}$. The other unknowns are computed through the back-substitution phase. Given $\mathbf{z}_i^{(k+1)}$, $\mathbf{z}_{i+s}^{(k+1)}$, in the $(p-k)$th step, for $k = p-1, \ldots, 1$, the unknowns $\mathbf{z}_i^{(k)}$, $\mathbf{z}_{i+s}^{(k)}$ are determined, using (3.17) and (3.21), from

$$\mathbf{z}_i^{(k)} \quad = \quad P_i^T \begin{pmatrix} (L_i U_i)^{-T} \mathbf{f}_i^{(k)} - {G_i^{(k)}}^T \mathbf{z}_{i+s}^{(k+1)} \\ \mathbf{z}_{i+s}^{(k+1)} \end{pmatrix}, \tag{3.25}$$

for $i = (2j-1)s$, $j = \lfloor \lceil N/s \rceil / 2 \rfloor, \ldots, 2, 1$, with $s = 2^k$. Note that, $z_0^{(k)} = z_0$ is satisfied for any k.

The following algorithm summarizes the operations performed on the right hand side $\mathbf{f}$ of the system (3.13), the reduction steps for the coefficient matrix are the same as executed in the factorization of A:

```
% Solution of the transposed linear system
 % Reduction of the right hand side
  p = ⌈log2(N)⌉
  for k = 0, ..., p − 1
     s = 2^k
     f_0^(k+1) = f_0^(k) − S̃_0^T (L_1 U_1)^{−T} f_1^(k)
     f_2^(k+1) = f_2^(k) − R̃_2^T (L_1 U_1)^{−T} f_i^(k)
     for j = 2, ..., ⌊⌈N/s⌉/2⌋
        i = (2j − 1)s
        f_{i−s}^(k+1) = f_{i−s}^(k+1) − S̃_{i−s}^T (L_i U_i)^{−T} f_i^(k)
        f_{i+s}^(k+1) = f_{i+s}^(k) − R̃_{i+s}^T (L_i U_i)^{−T} f_i^(k)
     end
  end
 % Solution of the final 2 × 2 block linear system
  ( z_0 ; z_N^(p) ) = ( D_a^T  S_0^(p)T ; D_b^T  R_N^(p)T )^{−1} P̃ ( f_0 ; f_N^(p) )
 % Back substitution
  for k = p − 1, ..., 0
     s = 2^k
     for j = ⌊⌈N/s⌉/2⌋, ..., 2, 1
        i = (2j − 1)s
        z_i^(k+1) = (L_i U_i)^{−T} f_i^(k) − G_i^(k)T z_{i+s}^(k+1)
        ( z_i^(k) ; z_{i+s}^(k) ) = P_i^T ( z_i^(k+1) ; z_{i+s}^(k+1) )
     end
  end
```

3.5 Description of the software

The *BABDCR* package is written in Fortran 90 and available in double precision. It has four main subroutines:

- BABDCR_FACT factorizes the BABD coefficient matrix in (3.1);
- BABDCR_SOLV solves the linear system (3.1) with a coefficient matrix factorized by BABDCR_ FACT;
- BABDCR_SOLVT solves the transposed linear system $A^T\mathbf{z} = \mathbf{f}$ with the coefficient matrix A factorized by BABDCR_FACT;
- BABDCR_FACTSOLV factorizes the BABD coefficient matrix and at the same time solves the linear system (3.1).

The last subroutine is convenient if only a single BABD linear system needs to be solved. The original coefficient matrix is replaced with part of the factorization and can no longer be used. On the other hand, the first two subroutines solve system (3.1) in two successive steps: BABDCR_FACT factorizes the coefficient matrix and BABDCR_SOLV uses the output of BABDCR_FACT to compute the solution of (3.1), the arrays containing the cyclic reduction factorization are not modified by successive calls to BABDCR_SOLV. Thus, the solution of q linear systems with the same coefficient matrix can be computed by means of one call to BABDCR_FACT followed by q calls to BABDCR_SOLV. This procedure yields a great decrease in the number of operations compared to the multiple use of BABDCR_FACTSOLV; however, it requires fill-in vectors (see below). Finally, if A is factorized with BABDCR_FACT, then BABDCR_SOLVT uses its output to solve the linear system $A^T\mathbf{z} = \mathbf{f}$. This subroutine is mainly used to compute the 1-norm of the inverse of the coefficient matrix in order to determine the condition number of the BABD coefficient matrix. This can be obtained by means of the subroutine DONEST [47], which needs in order to evaluate the 1-norm of the matrix A^{-1}: the product $A^{-1}\mathbf{x}$ (for example, computed by means of BABDCR_SOLV) and the product $(A^{-1})^T\mathbf{x}$ (for example, computed by means of BABDCR_SOLV) for a given vector $\mathbf{x}$.

The package requires that the coefficient matrix in input is given as in Figure 3.1, that is, the block rows

$$V_i := (S_{i-1}, R_i) \tag{3.26}$$

must be given sequentially in an $m \times m \times 2N$ three-dimensional array MATR_A; boundary blocks are saved in two $m \times m$ arrays LFTBLK and RGTBLK. The right hand side $\mathbf{f}$ must be assigned in an $m \times (N+1)$ array VECT_B.

The structure of BABDCR_FACT and BABDCR_SOLV is represented in Figures 3.2-3.3. Dashed blocks contain the Fortran 90 intrinsic procedure RESHAPE, the BLAS routines DGER, DTRSM, DGEMV and DDOT, and the Lapack routines DGETRF and DGETRS.

BABDCR_FACT involves calls to REASSEMBLE and REDUCE_BLOCK. The subroutine REASSEMBLE allows us to assemble a $2m \times m$ block from

$$\text{MATR\_A} = \left(\boxed{V_1} \; \boxed{V_2} \; \ldots \; \boxed{V_N} \right)$$

$$\text{LFTBLK} = \left(\boxed{D_a} \right) \qquad \text{RGTBLK} = \left(\boxed{D_b} \right)$$

Figure 3.1: Structure of the input coefficient matrix

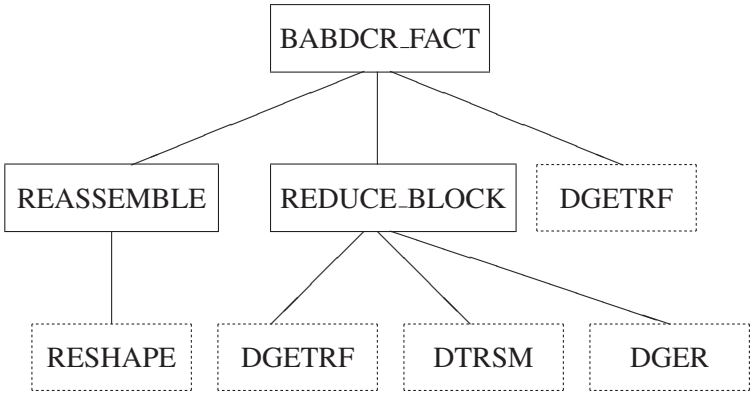

Figure 3.2: BABDCR_FACT subroutine hierarchy

two consecutive blocks of size $m \times m$. The output obtained is used by REDUCE_BLOCK which applies one step of reduction to obtain one block row from two consecutive ones (see (3.9)-(3.10)). The operations performed in REDUCE_BLOCK are summarized in Figure 3.5, where the variables considered have the same meaning of those in Section 3.1.

Since the reduction is also applied to the right hand side, we need to save, after each step of the coefficient matrix reduction, the block G_i which is used in REDUCE_RHS; moreover, we need to save the matrices L_i and U_i of the LU factorization of the $2m \times m$ block $\begin{pmatrix} R_i^{(k)} \\ S_i^{(k)} \end{pmatrix}$ (computed by the LAPACK routine DGETRF) and the fill-in block F_i of size m by m containing the nonzero rows of $\widehat{S}_{i-s}^{(k)}$ and $\widehat{R}_{i+s}^{(k)}$ which are used in SOLVE_BLOCK. Finally, $S_{i-s}^{(k)}$ and $R_{i+s}^{(k)}$ are replaced by the blocks of the new reduced matrix, namely $S_{i-s}^{(k+1)}$ and $R_{i+s}^{(k+1)}$.

On exit, BABDCR_FACT outputs two new arrays containing the fill-in blocks F_i and the permutations, saved in a $m \times m \times (N-1)$ array and in a $2m \times N$ integer array, respectively. The arrays containing the coefficient matrix now have the structure in Figure 3.5: the first and the last block of MATR_A and the boundary blocks contain the LU factorization of the matrix in (3.12) (see Figure 3.6), whereas each LUG_i block contains L_i, U_i and G_i (saved as in Figure 3.5) reshaped as a $2m \times m$ array.

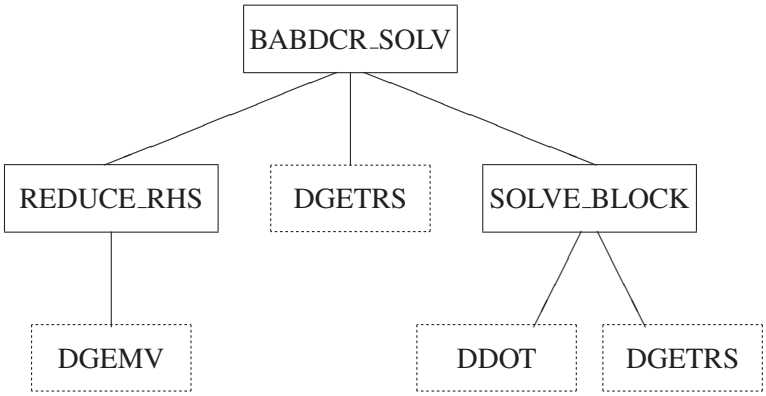

Figure 3.3: BABDCR__SOLV subroutine hierarchy

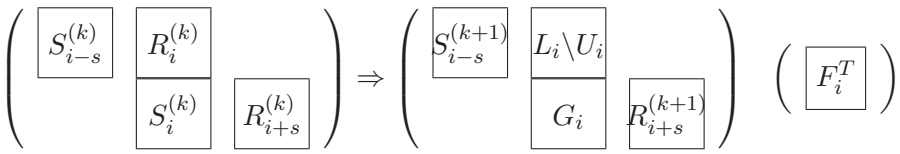

Figure 3.4: Operations performed in REDUCE__BLOCK

The subroutine BABDCR_SOLV uses the subroutines REDUCE_RHS to compute the right hand side of (3.10) from (3.9), and SOLVE_BLOCK to back-solve, that is, to compute x_s from (3.11) (x_{i-s} and x_{i+s} are known values). The LAPACK routine DGETRS solves the final 2×2 block linear system factorized by DGETRF.

The subroutine BABDCR_SOLVT uses the subroutine REDUCE_RHST to determine the right hand side of (3.22), it uses the subroutine DGETRS for solving linear systems and DAXPY for performing the multiplications in (3.23). The unknowns are determined in SOLVE_BLOCKT which performs the operations in (3.25) using the BLAS routine DGEMV. The routine DGETRS also performs the solution of the final 2×2 block linear system (3.24) factorized by DGETRF in BABDCR_FACT.

Finally, the subroutine BABDCR_FACTSOLV uses the subroutines REDUCE (which include the subroutines REDUCE_BLOCK and REDUCE_RHS), REASSEM-

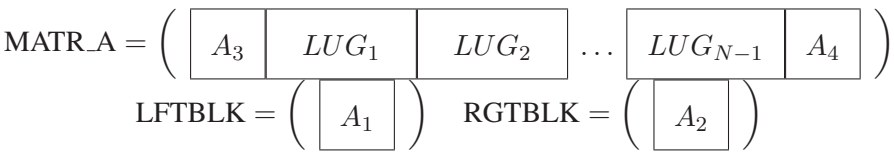

Figure 3.5: Structure of the coefficient matrix on exit from BABDCR__FACT

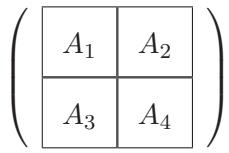

Figure 3.6: Structure of the LU factorization of (3.12)

BLE and SOLVE_BLOCK. Since the linear system is solved step by step with the factorization, the block G_i is used immediately after it is computed and so does not need to be saved. For this reason, the fill-in block F_i is saved in place of G_i and the subroutine essentially does not generate fill-in. The coefficient matrix is modified by the subroutine. It cannot be used for solving other systems with different right-hand sides, because the fill-in blocks are not saved and the permutation matrices are not output.

3.6 Calling sequences

We list the parameters used in the calling sequences of the four main subroutines of the package *BABDCR*. Note that, unless otherwise stated, the parameters do not change on exit.

The subroutine BABDCR_ FACT has the calling sequence:

CALL BABDCR_FACT(m, N, MATR_A, LFTBLK,
RGTBLK, PERM, FILL_IN, INFO)

where the parameters are defined as follows:

m integer, number of rows of each block V_i, see (3.26), in the coefficient matrix (3.1).

N integer, number of blocks V_i.

MATR_ A real, $m \times m \times 2N$ array. On input, it contains the coefficient matrix of the BABD system (3.1) which is stored as in Figure 3.1. On exit, it contains the factorization of the coefficient matrix which is stored as in Figure 3.5.

LFTBLK real, $m \times m$ array. On input, it contains the boundary block D_a. On exit, it contains the upper-left block A_1, see Figure 3.6, of the LU factorization of the final 2×2 block matrix in (3.12).

RGTBLK real, $m \times m$ array. On input, it contains the boundary block D_b. On exit, it contains the upper-left block A_2, see Figure 3.6, of the LU factorization of the final 2×2 block matrix in (3.12).

PERM integer, $2m \times N$ array. On exit, each column contains the permutation vector associated with the LU factorization performed by each call to REDUCE_BLOCK. The last two columns are associated with the permutation performed by the factorization of the final 2×2 block matrix in (3.12).

FILL_ IN real, $m \times m \times (N-1)$ array. On exit, it contains the fill-in blocks $F_i^{(k)}$ generated by the factorization (by each call to REDUCE_BLOCK).

INFO integer. On exit, this parameter is equal to 0 if the coefficient matrix is nonsingular, or gives the index[2] of the block where breakdown has occurred if A is singular.

The subroutine BABDCR_SOLV has the calling sequence:

```
CALL BABDCR_SOLV( m, N, MATR_A, LFTBLK,
                  RGTBLK, PERM, FILL_IN, VECT_B )
```

where the first seven parameters were defined in the subroutine BABDCR_FACT (and contain the output of that subroutine). The last parameter is

VECT_B real, $m \times (N+1)$ array. On input, it contains the right hand side $\mathbf{f}$ of system (3.1). On exit, it contains the solution of the system.

The subroutine BABDCR_SOLV has the calling sequence:

```
CALL BABDCR_SOLVT( m, N, MATR_A, LFTBLK
                   RGTBLK, PERM, FILL_IN, VECT_B )
```

As for BABDCR_SOLV, the first seven parameters were defined in the subroutine BABDCR_FACT (and contain the output of that subroutine). The last parameter is

VECT_B real, $m \times (N+1)$ array. On input, it contains the right hand side $\mathbf{f}$ of system $A^T\mathbf{z} = \mathbf{f}$, see (3.14). On exit, it contains the solution of the system.

Finally, the subroutine BABDCR_ FACTSOLV has the calling sequence:

```
CALL BABDCR_ FACTSOLV( m, N, MATR_ A, VECT_ B, INFO )
```

where the parameters are defined like those used in BABDCR_FACT. Since, in BABDCR_FACTSOLV, we do not heap the permutations array and the fill-in blocks, we cannot use the factorization on exit for solving more linear systems.

[2] If k (non-negative integer) and j (positive even integer between 2 and $N/2^k$) are such that INFO = $2^k(j-1)+1$, then the breakdown has occurred in the jth block column of the coefficient matrix of the linear system obtained after k steps of reduction.

3.6.1 A new version of *MIRKDC*

The code *MIRKDC*, whose algorithm is described in Section 1.3.1, uses the following variable names: `neqns` and `Nsub` are, respectively, the order m of the system (1.18) and the number N of subintervals of the current mesh, `leftbc` is the number of boundary conditions at the point a, `MxNsub` is the user defined maximum number of subintervals of $[a, b]$, and `blocks` is a vector which contains the blocks $V_i := (S_i, R_{i+1})$ of the current BABD coefficient matrix in (1.20), stored column by column, as for the array MATR_A in Figure 3.1.

We present a variant of *MIRKDC*, called *MIRKDC_2*, which solves the system of BVPs (1.18) with general non-separated boundary conditions. The algorithm uses the discretizations of *MIRKDC* resulting in linear systems (1.20) with the BABD form, as in (3.1), that are solved using the BABD solver *BABDCR*. Therefore, in *MIRKDC_2* we essentially replace *COLROW*, employed in *MIRKDC*, with *BABDCR*. To do this, we make some modifications of the *MIRKDC* code:

- the permutation vector array, of length `neqns*(MxNsub+1)` is replaced by a vector array of length `2*neqns*MxNsub`,
- the array FILL_IN, described in Section 3.5, is saved in `(Nsub-1)*(neqns**2)` contiguous locations in the array `blocks`,
- the arrays `top` and `bot` contain the blocks D_a and D_b of the Jacobian in (1.20), represented in (3.1), which each are of size `neqns`×`neqns` instead of `leftbc`×`neqns` and `(neqns-leftbc)`×`neqns`, respectively,
- in *BABDCR* the right hand side associated with the linear system is overwritten by the solution, whereas *COLROW* doesn't overwrite the solution. Therefore, before the call to *BABDCR* the right hand side occupies the same locations as for the solution.

3.7 Comparisons among the linear system solvers

We compare *BABDCR* with *COLROW* and *RSCALE* on the ABD and BABD linear systems generated by the codes *MIRKDC* and *MIRKDC_2*. Our tests are executed on an Alpha-server DS20E with a 667 MHz EV67 processor with a Compaq Fortran 90 (formerly Digital Fortran 90) compiler. Here and in the next section we use in *MIRKDC* and *MIRKDC_2* the optimal 3 stages, 4-th order MIRK scheme

$$\begin{array}{c|c|ccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{8} & -\frac{1}{8} & 0 \\ \hline & & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array}$$

and an associated 4-th order, 4 stages CMIRK scheme, see Section and [33]. First, we discuss the ABD linear systems generated by *MIRKDC* applied to a linear BVP

$\mathbf{y}' = M\mathbf{y}(t)$ with $M \in \mathbb{R}^{m\times m}$ and linear boundary conditions $D_{top}\mathbf{y}(a) = \mathbf{d}_a \in \mathbb{R}^{m_{top}}$ and $D_{bot}\mathbf{y}(b) = \mathbf{d}_b \in \mathbb{R}^{m-m_{top}}$.

Here we fix $m = 20$ and $m_{top} = 10$. Both D_{top}, D_{bot} and M are randomly generated full matrices and such that the BVP is well-conditioned. That is, $D_{top}, D_{bot} \in \mathbb{R}^{10\times 20}$ have rank 10, which guarantees the unicity of the solution. Also, matrix M is obtained from $M = Q\Lambda Q^T$, where Q is an orthogonal matrix arising from the QR factorization of a random matrix and Λ is a diagonal matrix with 10 positive and 10 negative diagonal values of moderate size. Then from the Proposition 1.1.10 and 1.1.14, the resulting BVP is well conditioned with an exponential dichotomy of rank 10.

BABDCR and *RSCALE* require the simple transformation

$$D_a = \begin{pmatrix} D_{top} \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times m}, \qquad D_b = \begin{pmatrix} 0 \\ D_{bot} \end{pmatrix} \in \mathbb{R}^{m\times m}$$
$$\text{and} \qquad \mathrm{d} = \begin{pmatrix} d_a \\ d_b \end{pmatrix} \in \mathbb{R}^{\mathrm{m}},$$

in order to be applied to a system with the BABD structure in (3.1). The timings and errors, in Table 3.1, lead us to prefer *COLROW* over the other algorithms. Indeed, *COLROW* is more than 2 times faster than *BABDCR* and more than 4 times faster than *RSCALE*. Moreover, the errors for *COLROW* and *BABDCR* are similar, but *RSCALE* is less accurate. These results essentially agree with the theoretical computational costs of the three solvers. In fact, applied to the ABD linear system in Fig. 2, to leading order in m, *COLROW* requires $\left(\frac{5}{3}m^3 + m(m-m_{top})m_{top}\right) N$ operations and *RSCALE* $\left(\frac{20}{3}m^3\right) N$.

Table 3.1: ABD systems generated by *MIRKDC* applied to a linear problem.

	time (secs.)			error		
	N=256	N=512	N=1024	N=256	N=512	N=1024
BABDCR	3.71e-02	7.12e-02	0.152	1.65e-13	2.88e-13	3.46e-13
COLROW	1.46e-02	2.83e-02	6.34e-02	1.55e-13	2.05e-13	7.08e-13
RSCALE	6.83e-02	0.136	0.274	2.54e-12	6.02e-12	3.01e-11

For a comparison on BABD linear systems, we apply *MIRKDC_2* to a linear BVP $\mathbf{y}' = M\mathbf{y}(t)$ with non-separated boundary conditions $D_a\mathbf{y}(a) + D_b\mathbf{y}(b) = \mathbf{d} \in \mathbb{R}^m$. Again, the size of the problem is $m = 20$. For M, we investigate two cases:

1. M is a well-conditioned matrix with eigenvalues -102, -10, -7, -4, -3, -2.5, -1.3, -1, -0.5, -0.4, 0.2, 0.3, 1, 1, 2, 2.5, 3, 4, 11, 25;

2. M has eigenvalues -9, -3.5, -3, -2, -2, -1.5, -1.5, -1.25, -0.5, -1e-08, 0.25, 0.5, 0.5 , 1, 3, 4, 5, 7, 8, 1e+08.

For *COLROW* we re-write the BABD system as an equivalent ABD linear system of double size, see 2.7. Then, the computational cost of this solver becomes $\left(\frac{46}{3}m^3\right) N$. This implies that theoretically *BABDCR* is more than three times faster. In Tables 3.2-3.3, we compare the errors and timings of the three linear solvers. From the results in Table 3.2 on the cases 1.-2., *BABDCR* is approximately 3 times faster than *COLROW* and more than 1.5 times faster than *RSCALE*. Timings associated with *COLROW* include converting the linear system from the BABD structure to the ABD structure 2.7. The errors associated with *BABDCR* and *COLROW* are similar, and *RSCALE* is the least accurate algorithm. The errors of the three methods, applied to case (2), are given in Table 3.3. Note that these errors are large, because the BVP is ill-conditioned for the presence of a null eigenvalue, see Proposition 1.1.9 and Theorem 1.1.10. Finally, observe that *BABDCR* and *COLROW* are significantly more accurate than *RSCALE*.

Table 3.2: Times in seconds for the solution of BABD systems generated by *MIRKDC_2* applied to a linear 20×20 BVP.

	N=256	N=512	N=1024
BABDCR	3.61e-02	7.03e-02	0.151
COLROW	0.102	0.224	0.464
RSCALE	6.83e-02	0.136	0.287

Table 3.3: Errors for the solution of BABD systems generated by *MIRKDC_2* applied to a linear 20×20 BVP.

	case (1)			case (2)		
	N=256	N=512	N=1024	N=256	N=512	N=1024
BABDCR	7.62e-13	1.22e-12	1.19e-12	6.28e-04	2.22e-04	8.00e-05
COLROW	7.53e-13	4.10e-13	1.25e-12	2.80e-04	2.16e-04	9.25e-05
RSCALE	5.17e-12	2.90e-11	6.02e-11	1.90e-02	8.55e-03	1.08 e-02

3.8 Comparisons of the two versions of *MIRKDC*

In order to better emphasize the advantages of using *BABDCR*, Tables 3.4-3.5 give statistics for calls to *MIRKDC_2* (that uses *BABDCR*) and to *MIRKDC* (using *COLROW*) applied to the BVP of double size $\mathbf{y}' = M\mathbf{y}(t)$, $\mathbf{z}' = \mathbf{0}$ with separated boundary conditions $\mathbf{y}(a) - \mathbf{z}(a) = \mathbf{0}$ and $D_a\mathbf{z}(b) + D_b\mathbf{y}(b) = \mathbf{d}$. Both codes are applied to the problem with M having eigenvalues as in case (1) using the MIRK/CMIRK scheme of order 4 [34]. From Tables 3.4 and 3.5, *BABDCR* in

Table 3.4: *MIRKDC_2* (using *BABDCR*) for the linear problem in case (1) with an initial mesh of 256 points and tolerance 1e-07

MESH	♯FACTs	time (secs.)	♯SOLVEs	time (secs.)
256	1	0.32e-01	2	0.98e-02
224	1	0.25e-01	2	0.78e-02
246	1	0.29e-01	2	0.78e-02
Total:	3	0.87e-01	6	0.25e-01
Total monitored Linear Algebra time:				0.11 secs.
Total monitored Nonlinear Algebra time:				0.12 secs.

Table 3.5: *MIRKDC* (using *COLROW*) for the linear problem of double size in case (1) with an initial mesh of 256 points and tolerance 1e-07.

MESH	♯FACTs	time (secs.)	♯SOLVEs	time (secs.)
256	1	0.11	2	0.16e-01
224	1	0.75e-01	2	0.12e-01
246	1	0.82e-01	2	0.14e-01
Total:	3	0.27	6	0.41e-01
Total monitored Linear Algebra time:				0.31 secs.
Total monitored Nonlinear Algebra time:				0.12 secs.

MIRKDC_2 saves more than one half of the linear algebra time with respect to using *COLROW*. Though this result is valid for the problem considered, results in Table 3.2 show that, for a general BVP with non-separated boundary conditions, *BABDCR* in *MIRKDC* runs significantly faster than *COLROW*.

Chapter 4

Algorithms to solve general BABD linear systems

In Section 1.7, we have determined the BABD structures of the linear systems arising from Orthogonal Spline Collocation (OSC) applied to a BVODE. In these structures, see (1.32) and (1.37), the block rows V_i have some columns not overlapped with other block rows. Also, the number of columns (or rows) in the blocks V_i may not be constant. Therefore the code *BABDCR*, described in the previous chapter, cannot be applied to these 'general' structures. We generalize the cyclic reduction approach, employed in *BABDCR*, for solving these BABD systems. Different cyclic reduction approaches are described. In particular, a new technique for systems arising from spline collocation with a monomial basis, as in (1.37), is introduced. We motivate the choice of an approach with respect to other approaches, analyzing times, errors and memory requirements. Comparisons with other codes are also provided.

4.1 Introduction

We consider the BABD linear system

$$A\mathbf{y} = \mathbf{f}, \tag{4.1}$$

with the coefficient matrix A having the 'general' structure

$$A = \begin{pmatrix} D_a & & & & & & & & D_b \\ V_1 & & & & & & & & \\ & V_2 & & & & & & & \\ & & V_3 & & & & & & \\ & & & V_4 & & & & & \\ & & & & \ddots & & & & \\ & & & & & \ddots & & & \\ & & & & & & V_N & & \end{pmatrix}.$$

Since each block row V_i has overlapping columns with the adjacent block rows, it may be represented with the following structure

$$V_i = \begin{pmatrix} S_{i-1} & T_i & R_i \end{pmatrix}, \qquad i = 1, \ldots, N, \tag{4.2}$$

where blocks R_{i-1} (in V_{i-1}) and S_{i-1} (in V_i) contain the overlapping columns between V_{i-1} and V_i. Therefore, the resulting structure may be represented as in the following

$$A = \begin{pmatrix} D_a & & & & & & & & & & D_b \\ S_0 & T_1 & R_1 & & & & & & & & \\ & & S_1 & T_2 & R_2 & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & S_{i-1} & T_i & R_i & & & \\ & & & & & & & \ddots & & & \\ & & & & & & & & \ddots & & \\ & & & & & & & & S_{N-1} & T_N & R_N \end{pmatrix}. \tag{4.3}$$

Note that, S_0 and R_N contain the same columns of the boundary blocks D_a and D_b, respectively. Also, T_i has no overlapping columns with any other block row. We set $n_i \times (m_{i-1} + k_i + m_i)$ the size of each block row V_i in (4.2), where n_i is the number of rows, m_i is the number of columns of R_i (and S_i in V_{i+1}), and k_i is the number of columns of T_i. Also, n_0 is the number of rows of the boundary blocks D_a and D_b. Then, supposing that the coefficient matrix A is square of dimension

$$N_{dim} = \sum_{i=0}^{N} n_i, \tag{4.4}$$

we obtain

$$\sum_{i=0}^{N} n_i = \sum_{i=0}^{N} m_i + \sum_{i=1}^{N} k_i,$$

and, for the nonsingularity of the linear system (4.1), we require

$$\begin{array}{ll} n_i \geq k_i, & i = 1, \ldots N, \\ n_i + n_{i+1} \geq k_i + m_i + k_{i+1}, & i = 1, \ldots, N-1. \end{array} \tag{4.5}$$

In the proof of conditions (4.5), we state that if $n_i + n_{i+1} < k_i + m_i + k_{i+1}$ is satisfied, then the submatrix

$$\begin{pmatrix} T_i & R_i & \\ & S_i & T_{i+1} \end{pmatrix} \tag{4.6}$$

of size $N_{dim} \times (k_i + m_i + k_{i+1})$ extracted from (4.3), has $k_i + m_i + k_{i+1}$ linear independent columns. Hence, (4.6) has more than $n_i + n_{i+1}$ linear independent rows. This reductio ad absurdum that some null rows are linear independent. Analogously, the condition $n_i \geq k_i$ is satisfied because the k_i (no overlapping) columns of T_i are linear independent.

As an example, see Section 1.7, the BABD structure (4.3) arises from the numerical solution of two-point boundary value problems with non-separated boundary conditions, as in (1.4). Considering a BVP of order m and using k Gaussian points in each of the N subintervals, the resulting BABD structure has block rows (4.2) with

- in the B-spline case (Section 1.7.1):

$$S_{i-1} = W_{i,1}, \quad T_i = W_{i,2}, \qquad R_i = W_{i,3}, \tag{4.7}$$

  with $W_{i,1}$, $W_{i,3} \in \mathbb{R}^{k \times m}$ and $W_{i,2} \in \mathbb{R}^{k \times (k-m)}$,

- in the monomial spline case (Section 1.7.2):

$$S_{i-1} = \begin{pmatrix} H_i \\ -E_i \end{pmatrix}, \quad T_i = \begin{pmatrix} G_i \\ -F_i \end{pmatrix} \quad \text{and} \quad R_i = \begin{pmatrix} 0 \\ I \end{pmatrix}, \tag{4.8}$$

  where $-E_i \in \mathbb{R}^{m \times m}$, the identity matrix $I \in \mathbb{R}^{m \times m}$, $H_i \in \mathbb{R}^{k \times m}$, $-F_i \in \mathbb{R}^{m \times k}$, and $G_i \in \mathbb{R}^{k \times k}$.

We use the following notation for the right hand side $\mathbf{f}$ and for the unknown vector $\mathbf{y}$ of system (4.1),

$$\begin{aligned} \mathbf{f} &= \left(\begin{array}{cccc} \mathbf{d}^T, & \mathbf{f}_1^T, & \ldots, & \mathbf{f}_N^T \end{array} \right)^T, \\ \mathbf{y} &= \left(\begin{array}{cccccccc} \mathbf{z}_0^T, & \mathbf{w}_1^T, & \mathbf{z}_1^T, & \mathbf{w}_2^T, & \ldots, & \mathbf{z}_{N-1}^T, & \mathbf{w}_N^T, & \mathbf{z}_N^T \end{array} \right)^T, \end{aligned} \tag{4.9}$$

where $\mathbf{d} \in \mathbb{R}^{n_0}$, $\mathbf{f}_i \in \mathbb{R}^{n_i}$, $\mathbf{z}_i \in \mathbb{R}^{m_i}$, and $\mathbf{w}_i \in \mathbb{R}^{k_i}$.
We note that, in the numerical discretization of BVODEs, using the techniques described in Chapter 1, the number of block rows N is usually much larger than n_i, m_i, and k_i.

4.2 Cyclic reduction approach for general BABD linear systems

The cyclic reduction algorithm, described in Chapter 3, can be improved to solve BABD linear systems (4.1) with structure (4.3). This algorithm yields systems of lower dimension (involving less unknowns) with the same BABD structure, until it reaches a small system that is easy to solve directly. After that, using back-substitution, all the unknowns of the system are determined.

At the first step, for $i = 2j-1,\ j = 1, 2, \ldots, \lfloor N/2 \rfloor$, each subsystem

$$\begin{aligned} S_{i-1}\mathbf{z}_{i-1} + T_i\mathbf{w}_i + R_i\mathbf{z}_i &= \mathbf{f}_i \\ S_i\mathbf{z}_i + T_{i+1}\mathbf{w}_{i+1} + R_{i+1}\mathbf{z}_{i+1} &= \mathbf{f}_{i+1} \end{aligned} \tag{4.10}$$

involving $s_i = m_{i-1} + k_i + m_i + k_{i+1} + m_{i+1}$ unknowns, is transformed in the equation

$$S'_{i-1}\mathbf{z}_{i-1} + T'_i\mathbf{w}'_i + R'_{i+1}\mathbf{z}_{i+1} = \mathbf{f}'_{i+1}, \tag{4.11}$$

involving only the $s_i - r_i$ (with $r_i > 0$) unknowns: $\mathbf{z}_{i-1}$, $\mathbf{z}_{i+1}$ and $\mathbf{w}'_i$, described below.

Note that, the blocks S'_{i-1} and R'_{i+1} correspond to the same unknowns associated to S_{i-1} and R_{i+1}, respectively. Therefore, after the first step of the algorithm, the linear system

$$\begin{pmatrix} D_a & & & & & & D_b \\ S'_0 & T'_1 & R'_2 & & & & \\ & & S'_2 & T'_3 & R'_4 & & \\ & & & & \ddots & & \\ & & & & S'_{N-2} & T'_{N-1} & R'_N \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{w}'_1 \\ \mathbf{z}_2 \\ \mathbf{w}'_3 \\ \vdots \\ \mathbf{z}_{N-2} \\ \mathbf{w}'_{N-1} \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}'_2 \\ \mathbf{f}'_4 \\ \vdots \\ \mathbf{f}'_N \end{pmatrix} \tag{4.12}$$

is obtained. The blocks R'_{i-1} and S'_{i-1} lie in the same columns of the BABD coefficient matrix in (4.12). Also, S'_0 contains the overlapping columns of the first block row with D_a and R'_N contains the overlapping columns of the last block row with D_b. If N is odd, the last row of the initial system with coefficient matrix (4.3) is unchanged, that is

$$S'_{N-1} = S_{N-1}, \qquad T'_N = T_N, \qquad R'_N = R_N.$$

Therefore, the number of block rows of the resulting linear system (4.12) is $\lceil N/2 \rceil + 1$.

There are several ways to reduce subsystems (4.10) to (4.11); at least four as represented in Figure 4.1. In fact, since each matrix (4.6) has $k_i + m_i + k_{i+1}$ linear independent columns, any submatrix M_i, in Figure 4.1, has rank $r_i \leq n_i + n_{i+1}$.

1. $M_i = \begin{pmatrix} T_i & R_i & & \\ & S_i & T_{i+1} & \end{pmatrix}$, $r_i = k_i + m_i + k_{i+1}$, yields to a null vector $\mathbf{w}'_i$, because T'_i is not determined;
2. $M_i = \begin{pmatrix} R_i & & \\ S_i & T_{i+1} & \end{pmatrix}$, $r_i = m_i + k_{i+1}$, yields to $\mathbf{w}'_i = \mathbf{w}_i$, $T'_i \in \mathbb{R}^{(n_i+n_{i+1}-r_i)\times k_i}$;
3. $M_i = \begin{pmatrix} T_i & R_i \\ & S_i \end{pmatrix}$, $r_i = k_i + m_i$, yields to $\mathbf{w}'_i = \mathbf{w}_{i+1}$, $T'_i \in \mathbb{R}^{(n_i+n_{i+1}-r_i)\times k_{i+1}}$;
4. $M_i = \begin{pmatrix} R_i \\ S_i \end{pmatrix}$, $r_i = m_i$ leads to $\mathbf{w}'_i = \begin{pmatrix} \mathbf{w}_i \\ \mathbf{w}_{i+1} \end{pmatrix}$, $T'_i \in \mathbb{R}^{(n_i+n_{i+1}-r_i)\times(k_i+k_{i+1})}$.

Figure 4.1: M_i, r_i and $\mathbf{w}'_i$ used in four possible reduction of (4.10) in (4.11)

Thus, using the inverse of an $r_i \times r_i$ matrix extracted from M_i, we can express explicitly r_i unknowns of the subsystem (4.10) as a function of the remaining $s_i - r_i$. In the equation (4.11), the unknowns $\mathbf{w}'_i$ are those used in (4.10) and different from $\mathbf{z}_{i-1}$ and $\mathbf{z}_{i+1}$. The unknowns $\mathbf{w}'_i$ are of length $s_i - (r_i + m_i + m_{i+1}) = k_i + m_i + k_{i+1} - r_i$.

The reduction of any subsystem (4.10) is performed by row partial pivoting LU factorization applied to M_i:

$$P_i M_i = \begin{pmatrix} L_i \\ H_i \end{pmatrix} U_i = \begin{pmatrix} I & \\ G_i & I \end{pmatrix} L_i U_i.$$

The permutation matrix P_i is of size $(n_i + n_{i+1}) \times (n_i + n_{i+1})$, L_i and U_i are respectively lower and upper triangular matrices of dimension r_i, H_i is of size $(n_i + n_{i+1} - r_i) \times r_i$ and $G_i = H_i L_i^{-1} \in \mathbb{R}^{(n_i+n_{i+1}-r_i)\times r_i}$. The reduced system (4.12) is obtained by multiplying each pair of block rows (4.10) by P_i and by

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix},$$

the inverse of the lower triangular matrix $\begin{pmatrix} I & \\ G_i & I \end{pmatrix}$.
Then, in case 4 (Figure 4.1), the reduction yields

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} S_{i-1} & T_i & R_i & & \\ & & S_i & T_{i+1} & R_{i+1} \end{pmatrix} = \begin{pmatrix} \tilde{S}_{i-1} & \tilde{T}_i & L_i U_i & \tilde{T}_{i+1} & \tilde{R}_{i+1} \\ S'_{i-1} & T'_{i,1} & & T'_{i,2} & R'_{i+1} \end{pmatrix}. \tag{4.13}$$

The block T_i' in (4.11) is

$$T_i' = \begin{pmatrix} T_{i,1}' & T_{i,2}' \end{pmatrix}.$$

For the other cases in Figure 4.1, since T_i and/or T_{i+1} are in M_i, they are not involved in the operations for determining T_i'.

The right hand side is modified accordingly

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{f}}_i \\ \mathbf{f}_{i+1}' \end{pmatrix}. \tag{4.14}$$

From (4.13), it follows that the blocks S_{i-1}' and R_{i+1}' satisfy

$$\begin{aligned} S_{i-1}' &= \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} S_{i-1} \\ O \end{pmatrix}, \\ R_{i+1}' &= \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} O \\ R_{i+1} \end{pmatrix}, \end{aligned} \tag{4.15}$$

where S_{i-1}' is of size $(n_i + n_{i+1} - r_i) \times m_{i-1}$, and R_{i+1}' is of size $(n_i + n_{i+1} - r_i) \times m_{i+1}$.

In any of the four cases represented in Figure 4.1, the blocks S_{i-1}' and R_{i+1}' are determined through the operations in (4.15). In contrast, the block T_i' of size $(n_i + n_{i+1} - r_i) \times (k_i + m_i + k_{i+1} - r_i)$ differs in each case:

1. T_i' does not exist,
2. $T_i' = \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} T_i \\ O \end{pmatrix}$,
3. $T_i' = \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} O \\ T_{i+1} \end{pmatrix}$,
4. $T_i' = \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} T_i & O \\ O & T_{i+1} \end{pmatrix}$.

The right hand side $\mathbf{f}_{i+1}'$ is of length $n_i + n_{i+1} - r_i$ and satisfies

$$\mathbf{f}_{i+1}' = \begin{pmatrix} -G_i & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_i \\ \mathbf{f}_{i+1} \end{pmatrix}.$$

The reduced system (4.12) does not involve at least the unknowns $\mathbf{z}_{2j-1}$, for $j = 1, 2, \ldots, \lfloor N/2 \rfloor$. Since each $\mathbf{z}_{2j-1}$ is of length m_{2j-1}, the dimension N_{dim}, defined (4.4), of the initial system is reduced by

$$\sum_{j=1}^{\lfloor N/2 \rfloor} r_{2j-1}, \tag{4.16}$$

which is larger than

$$\sum_{j=1}^{\lfloor N/2 \rfloor} m_{2j-1}.$$

In case 4, the reduced system (4.12) is of dimension $N_{dim} - \sum_{j=1}^{\lfloor N/2 \rfloor} m_{2j-1}$, while in the other cases the dimension is much smaller.

Let us describe what happen in the next steps of cyclic reduction. Let $s = 2^k$, after k steps of reduction the coefficient matrix has $\lceil N/s \rceil + 1$ block rows. The linear system can be further reduced by combining two successive block row equations (for each $i = (2j-1)s$, with $j = 1, 2, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor$)

$$S_{i-s}^{(k)} \mathbf{z}_{i-s} + T_i^{(k)} \mathbf{w}_i^{(k)} + R_i^{(k)} \mathbf{z}_i = \mathbf{f}_i^{(k)}$$

$$S_i^{(k)} \mathbf{z}_i + T_{i+s}^{(k)} \mathbf{w}_{i+s}^{(k)} + R_{i+s}^{(k)} \mathbf{z}_{i+s} = \mathbf{f}_{i+s}^{(k)},$$

to give

$$S_{i-s}^{(k+1)} \mathbf{z}_{i-s} + T_i^{(k+1)} \mathbf{w}_i^{(k+1)} + R_{i+s}^{(k+1)} \mathbf{z}_{i+s} = \mathbf{f}_{i+s}^{(k+1)},$$

where

$$S_{i-s}^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_i^{(k)} \begin{pmatrix} S_{i-s}^{(k)} \\ O \end{pmatrix},$$

$$R_{i+s}^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_i^{(k)} \begin{pmatrix} O \\ R_{i+s}^{(k)} \end{pmatrix}.$$

In the cases in Figure 4.1, we have

1. $T_i^{(k+1)}$ does not exist,
2. $T_i^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_i^{(k)} \begin{pmatrix} T_i^{(k)} \\ O \end{pmatrix}$,
3. $T_i^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_i^{(k)} \begin{pmatrix} O \\ T_{i+s}^{(k)} \end{pmatrix}$,
4. $T_i^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_{i-s}^{(k)} \begin{pmatrix} T_i^{(k)} & O \\ O & T_{i+s}^{(k)} \end{pmatrix}$,

and

$$\mathbf{f}_{i+s}^{(k+1)} = \begin{pmatrix} -G_i^{(k)} & I \end{pmatrix} P_i^{(k)} \begin{pmatrix} \mathbf{f}_i^{(k)} \\ \mathbf{f}_{i+s}^{(k)} \end{pmatrix},$$

with $G_i^{(k)} = H_i^{(k)} \left(L_i^{(k)}\right)^{-1}$ and

$$P_i^{(k)} M_i^{(k)} = \begin{pmatrix} L_i^{(k)} \\ H_i^{(k)} \end{pmatrix} U_i^{(k)} = \begin{pmatrix} I & \\ G_i^{(k)} & I \end{pmatrix} \begin{pmatrix} L_i^{(k)} U_i^{(k)} \\ O \end{pmatrix},$$

where $M_i^{(k)}$ is one of the possible choices as shown in Figure 4.1.

After $p = \lceil \log_2(N) \rceil$ reduction steps, we obtain the system

$$\begin{pmatrix} D_a & & D_b \\ S_0^{(p)} & T_{\frac{N}{2}}^{(p)} & R_N^{(p)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{w}_{\frac{N}{2}}^{(p)} \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_N^{(p)} \end{pmatrix}. \tag{4.17}$$

The algorithm proceeds with the solution of (4.17) using Gaussian elimination, obtaining $\mathbf{z}_0$, $\mathbf{w}_{\frac{N}{2}}^{(p)}$ and $\mathbf{z}_N$. The other unknowns are computed through the back-substitution phase in a similar manner to that described in Chapter 3.

The algorithm has been described above in the most possible general way, using mainly case 4 as an example. However the approach used in case 4, is not appropriate, because it determines blocks T_i' of a larger size than the other cases. Also, the computational cost is larger. For example in case 4, the block $T_{\frac{N}{2}}^{(p)}$ is of size $(m_0 + m_N + \sum_{i=1}^{N} k_i - n_0) \times (\sum_{i=1}^{N} k_i)$. This yields a system (4.17) of a large dimension $(m_0 + m_N + \sum_{i=1}^{N} k_i)$. In contrast, in the first step of case 1, we obtain a matrix having no T_i' block and then as in *BABDCR* in the successive steps only $\begin{pmatrix} R_i \\ S_i \end{pmatrix}$ is factored. The resulting final system is

$$\begin{pmatrix} D_a & D_b \\ S_0^{(p)} & R_N^{(p)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_N^{(p)} \end{pmatrix},$$

it is of dimension $m_0 + m_N$. Therefore, the approach in case 4 is never used. We have described it in detail, because it describes cyclic reduction with more generality than the others.

In the next sections, we describe algorithms using cyclic reduction as described in cases 1,2 or 3, which solve linear systems arising from OSC. We show that the efficiency of these algorithms depends essentially on the BABD structure. In particular we optimize the algorithm associated with case 1. The resulting algorithm can easily be implemented also for solving systems with the general structure (4.3).

4.3 An algorithm for BABD systems arising from OSC

As seen in Section 4.1, the discretization of the mth ordinary differential equation (1.4) on $[a, b]$, using orthogonal spline collocation (OSC) with B-splines or

monomial splines yields a BABD system

$$\begin{pmatrix} D_a & & & & & & & D_b \\ S_0 & T_1 & R_1 & & & & & \\ & & S_1 & T_2 & R_2 & & & \\ & & & & \ddots & & & \\ & & & & & S_{N-1} & T_N & R_N \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{w}_1 \\ \mathbf{z}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{z}_{N-1} \\ \mathbf{w}_N \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}. \tag{4.18}$$

In (4.18), $S_i \in \mathbb{R}^{(\tilde{k}+m)\times m}$, $T_i \in \mathbb{R}^{(\tilde{k}+m)\times \tilde{k}}$, $C_i \in \mathbb{R}^{(\tilde{k}+m)\times m}$ and $D_a, D_b \in \mathbb{R}^{m\times m}$. If k is the number of Gaussian points in each subintervals used in the discretization of the ODE, the parameter $\tilde{k}$ used in the dimensions of the blocks above, is determined as follows:

- B-spline case, see (4.7),
$$\tilde{k} = k - m > 0,$$
- Monomial spline case, see (4.8),
$$\tilde{k} = k.$$

First, we apply the first step of the reduction in case 1 (Figure 4.1), that is, we employ Gaussian elimination to each matrix

$$M_i = \begin{pmatrix} T_i & R_i & \\ & S_i & T_{i+1} \end{pmatrix}, \qquad i = 2j-1, \text{ for } j = 1, \ldots, \lfloor N/2 \rfloor. \tag{4.19}$$

This yields a BABD system

$$\begin{pmatrix} D_a & & & & D_b \\ S_0' & R_2' & & & \\ & S_2' & R_4' & & \\ & & \ddots & \ddots & \\ & & & S_{N-2}' & R_N' \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_{N-2} \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_2' \\ \mathbf{f}_4' \\ \vdots \\ \mathbf{f}_N' \end{pmatrix}, \tag{4.20}$$

where $S_i', R_{i+2}' \in \mathbb{R}^{m\times m}$, for any $i = 0, 2, \ldots, N-2$.

The computational cost of this first step is

$$\lfloor N/2 \rfloor \left(\tfrac{14}{3}m^3 + 16\tilde{k}^2 m + 16\tilde{k}m^2 + \tfrac{16}{3}\tilde{k}^3 - 4\tilde{k}m - \tfrac{3}{2}m^2 - 2\tilde{k}^2 - \tfrac{1}{3}\tilde{k} - \tfrac{1}{6}m \right). \tag{4.21}$$

In the rth step, $r = 2, \ldots, p = \lceil \log_2(N) \rceil$, we apply Gaussian elimination to the matrix

$$M_i^{(r-1)} = \begin{pmatrix} R_i^{(r-1)} \\ S_i^{(r-1)} \end{pmatrix}, \qquad i = (2j-1)s, \text{ for } j = 1, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor,$$

where $s = 2^{r-1}$. The computational cost of these $p-1$ steps is

$$\lceil N/2 \rceil \left(\frac{14}{3}m^3 - \frac{3}{2}m^2 - \frac{1}{6}m \right) \tag{4.22}$$

We call this algorithm *GBABDCR-0*. From (4.21) and (4.22), the computational cost of the factorization phase is

$$N \left(\frac{8}{3}\tilde{k}^3 + 8\tilde{k}^2 m + +8\tilde{k}m^2 + \frac{14}{3}m^3 - \tilde{k}^2 - 2\tilde{k}m - \frac{3}{2}m^2 - \frac{1}{6}\tilde{k} - \frac{1}{6}m \right). \tag{4.23}$$

Note that, in the last $p-1$ steps of *GBABDCR-0*, the system can be solved using the algorithm *BABDCR*, described in Chapter 3.

4.4 *GBABDCR*: an optimized implementation of the cyclic reduction

The reduction process implemented in *GBABDCR-0* can be optimized by taking into account that the matrix (4.19) is not full and also that the unknowns $\mathbf{w}_i$ are only multiplied by T_i. As, we use a condensation step (see [10]) in order to eliminate the dependence on $\mathbf{w}_i$ in V_i. This step may be viewed as the first reduction step in the algorithm that eliminates the even unknowns $\mathbf{w}_i$ of the solution vector $\mathbf{y}$. We observe that blocks T_i of size $(\tilde{k}+m) \times \tilde{k}$, have full rank $\tilde{k}$ because they have no overlapping columns with other block rows. Then, we determine the factorization:

$$\widetilde{P}_i T_i = \begin{pmatrix} \widetilde{L}_i \\ \widetilde{H}_i \end{pmatrix} \widetilde{U}_i = \begin{pmatrix} I & \\ \widetilde{G}_i & I \end{pmatrix} \begin{pmatrix} \widetilde{L}_i \widetilde{U}_i \\ O \end{pmatrix}, \tag{4.24}$$

where $\widetilde{L}_i$ and $\widetilde{U}_i$ are square matrices of dimension $\tilde{k}$ and $\widetilde{G}_i = \widetilde{H}_i \widetilde{L}_i^{-1} \in \mathbb{R}^{m \times \tilde{k}}$.

Multiplying on the left V_i by $\widetilde{P}_i$ and the inverse of the lower triangular matrix in the last term of (4.24), we obtain

$$\begin{pmatrix} I & \\ -\widetilde{G}_i & I \end{pmatrix} \widetilde{P}_i \begin{pmatrix} S_{i-1} & T_i & R_i \end{pmatrix} = \begin{pmatrix} \widetilde{S}_{i-1} & \widetilde{L}_i \widetilde{U}_i & \widetilde{R}_i \\ \widehat{S}_{i-1} & & \widehat{R}_i \end{pmatrix}. \tag{4.25}$$

Analogously, we perform these operations on the right hand side $\mathbf{f}_i$ to obtain the vectors $\begin{pmatrix} \widetilde{\mathbf{f}}_i \\ \widehat{\mathbf{f}}_i \end{pmatrix}$. The row with the boundary blocks and the second row of (4.25), for $i = 2j-1,\ j = 1, 2, \ldots, \lfloor N/2 \rfloor$, yields the linear system

$$\begin{pmatrix} D_a & & & & D_b \\ \widehat{S}_0 & \widehat{R}_1 & & & \\ & \widehat{S}_1 & \widehat{R}_2 & & \\ & & \ddots & \ddots & \\ & & & \widehat{S}_{N-1} & \widehat{R}_N \end{pmatrix} \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \widehat{\mathbf{f}}_1 \\ \widehat{\mathbf{f}}_2 \\ \vdots \\ \widehat{\mathbf{f}}_N \end{pmatrix}. \tag{4.26}$$

The system (4.26) has dimension $(N+1)m$ and no longer depends on the unknowns $\mathbf{w}_i$. These unknowns will be computed in the last step of the cyclic reduction back-substitution phase (when all the $\mathbf{z}_i$ have already been computed) by using the first row of (4.25)

$$\widetilde{L}_i\widetilde{U}_i\mathbf{w}_i = \tilde{\mathbf{f}}_i - \widetilde{S}_{i-1}\mathbf{z}_{i-1} - \widetilde{R}_i\mathbf{z}_i.$$

Since $\widetilde{G}_i$ may be saved with $\widetilde{L}_i$ and $\widetilde{U}_i$ in place of T_i in the row block

$$\begin{pmatrix} S_{i-1} & T_i & R_i \end{pmatrix},$$

the factorization (4.25) does not require additional memory. Therefore, this condensation phase can be considered as a (completely parallelizable) initial step applicable to BABD systems, in order to simplify their structure before factorization.

The system (4.26) can be further reduced by using the algorithm *BABDCR*. When the LU factorization

$$P_i\begin{pmatrix} \widehat{R}_i \\ \widehat{S}_i \end{pmatrix} = \begin{pmatrix} L_i \\ H_i \end{pmatrix} U_i = \begin{pmatrix} I & \\ G_i & I \end{pmatrix}\begin{pmatrix} L_iU_i \\ O \end{pmatrix}$$

is applied to the block rows of index i and $i+1$ in (4.26), we have

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} \widehat{S}_{i-1} & \widehat{R}_i & \\ & \widehat{S}_i & \widehat{R}_{i+1} \end{pmatrix} = \begin{pmatrix} \bar{S}_{i-1} & L_iU_i & \bar{R}_i \\ S'_{i-1} & & R'_i \end{pmatrix}. \tag{4.27}$$

The first row may be used to compute $\mathbf{z}_i$ from $\mathbf{z}_{i-1}$ and $\mathbf{z}_{i+1}$.

This last factorization requires additional memory to store the fill-in blocks $G_i = H_iL_i^{-1}$. In fact G_i is used in the reduction of the right hand side of (4.26) corresponding to the block rows i and $i+1$. After this step, we obtain a BABD system with the same structure of (4.20) but with a number of operations

$$N(\tfrac{2}{3}\tilde{k}^3 + 2\tilde{k}^2m + 4\tilde{k}m^2 - \tilde{k}m - \tfrac{1}{2}\tilde{k}^2 - \tfrac{1}{6}\tilde{k}) + \lfloor N/2 \rfloor(\tfrac{14}{3}m^3 - \tfrac{3}{2}m^2 - \tfrac{1}{6}m). \tag{4.28}$$

Hence, we save

$$N(2\tilde{k}^3 + 6\tilde{k}^2m + 4\tilde{k}m^2 - \frac{1}{2}\tilde{k}^2 - \tilde{k}m) \tag{4.29}$$

operations with respect to *GBABDCR-0*.

As in *BABDCR*, iterating the last step on the successively reduced systems, we obtain, after $\lceil \log_2 N \rceil$ steps, a 2×2 block linear system

$$\begin{pmatrix} D_a & D_b \\ S_0^* & R_N^* \end{pmatrix}\begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0^* \\ \mathbf{f}_N^* \end{pmatrix}. \tag{4.30}$$

The solution of (4.30) with Gaussian elimination determines $\mathbf{z}_0$ and $\mathbf{z}_N$. Successively, a back-substitution phase allows to compute all the other unknowns. We call this optimized algorithm *GBABDCR*.

The computational cost of *GBABDCR*, in the factorization phase, is

$$N(\tfrac{2}{3}\tilde{k}^3 + 2\tilde{k}^2 m + 4\tilde{k}m^2 + \tfrac{14}{3}m^3 - \tfrac{1}{2}\tilde{k}^2 - \tilde{k}m - \tfrac{3}{2}m^2 - \tfrac{1}{6}\tilde{k} - \tfrac{1}{6}m). \quad (4.31)$$

Since *GBABDCR* proceeds as *GBABDCR-0* after the first two steps described above, the total saving in the computational cost of the factorization phase of *GBABDCR* is again (4.29).

4.4.1 Description of the software

The *GBABDCR* package has two main subroutines:

- GBABDCR_FACT performs the factorization of the coefficient matrix of the system (4.18);
- GBABDCR_SOLV uses the output of GBABDCR_ FACT to solve (4.18).

The solution of a set of p linear systems with the same coefficient matrix can be obtained by means of one call to GBABDCR_ FACT followed by p calls to GBABDCR_SOLV with a large saving in computational cost. In Figure 4.4.1, the calling sequences of the two subroutines are shown.

SUBROUTINE GBABDCR_FACT($\tilde{k}$, m, N, MATR_A, LFTBLK,
RGTBLK, PERM, FILL_IN, INFO)

SUBROUTINE GBABDCR_SOLV($\tilde{k}$, m, N, MATR_A, LFTBLK,
RGTBLK, PERM, FILL_IN, VECT_B)

Figure 4.2: Calling sequences of the two main subroutines

The package requires that the coefficient matrix in input is given as in Figure 4.3, that is, blocks (S_{i-1}, T_i, R_{i+1}) must be given sequentially and stored respectively in MATR_A which is a three-dimensional array of size $(\tilde{k}+m)\times(\tilde{k}+2m)\times N$.
The right hand side of (4.35) must be given in a vector VECT_B of length $(\tilde{k}+m)N+m$, see Figure 4.3. The boundary blocks D_a and D_b are saved, respectively, in the arrays LFTBLK and RGTBLK both of size $m \times m$.

The array FILL_IN is of size $m \times m \times (N-1)$ and contains the fill-in blocks generated by the factorizations. PERM is an integer array containing the permutation vectors associated with the LU factorizations.

$$\text{MATR\_A}= \left(\boxed{S_0} \boxed{T_1} \boxed{R_1} \ \ldots\ldots \ \boxed{S_{N-1}} \boxed{T_N} \boxed{R_N} \right),$$

$$\text{LFTBLK}= \left(\boxed{D_a} \right) \qquad \text{RGTBLK} = \left(\boxed{D_b} \right),$$

$$\text{VECT\_B}= \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}.$$

Figure 4.3: Structure of the input coefficient matrix and right hand side

Several algorithms are proposed in [1] for solving ABD (or BABD) systems arising from OSC. As for *GBABDCR*, some of these algorithms solve only particular systems in order to become more efficient than those written for general systems. For example *ABDPACK*, described in Section 2.5.1, solves only ABD systems (2.14) arising from OSC with monomial spline basis. In contrast, *COLROW* solves ABD systems with the same 'general' structure (4.18) used in *GBABDCR*. In the following sections, we compare timings and accuracy of our algorithms with respect to *COLROW* and *ABDPACK*.

4.5 Numerical comparisons

First, we compare the two algorithm *GBABDCR* and *GBABDCR-0*, based on the cyclic reduction approaches proposed above. We want to prove, rather than simply theoretically (*GBABDCR* requires less computational cost and storage), that condensation may reduce the execution time.

Second, we compare these codes with *COLROW* [29], which is one of the fastest available codes for ABD linear systems with coefficient matrices in the form

$$A = \begin{pmatrix} D_{top} & & & & & & & & \\ S_0 & T_1 & R_1 & & & & & & \\ & & S_1 & T_2 & R_2 & & & & \\ & & & & & & & & \\ & & & & \ddots & & & & \\ & & & & & & & & \\ & & & & & S_{N-1} & T_N & R_N & \\ & & & & & & & D_{bot} & \end{pmatrix}, \tag{4.32}$$

where $S_i, R_i \in \mathbb{R}^{(\tilde{k}+m)\times m}$, $T_i \in \mathbb{R}^{(\tilde{k}+m)\times \tilde{k}}$, $D_{top} \in \mathbb{R}^{m_{top}\times m}$, $D_{bot} \in \mathbb{R}^{(m-m_{top})\times m}$.

As discussed in Section 2.5, *COLROW* uses a modified alternate row and column elimination that generates no fill-in. When applied to (4.32), the computa-

tional cost of the factorization phase is

$$
\begin{array}{l}
N(\frac{2}{3}\tilde{k}^3 + 3\tilde{k}^2 m - \tilde{k}^2 m_{top} - \tilde{k} m_{top}^2 + 4\tilde{k}m^2 + \frac{5}{3}m^3 + m^2 m_{top} - m m_{top}^2 \\
\quad -\frac{1}{2}\tilde{k}^2 - 2\tilde{k}m - \frac{3}{2}m^2 + m m_{top} - m_{top}^2 - \frac{1}{6}\tilde{k} - \frac{1}{6}m) \\
+\frac{2}{3}m^3 + 2\tilde{k}m m_{top} + 2m m_{top}^2 - \tilde{k} m_{top}^2 - m_{top}^3 - \frac{1}{2}m^2 - \tilde{k} m_{top} - m_{top}^2 - \frac{1}{6}m.
\end{array}
$$

As discussed in Section 2.2, *COLROW* can solve the BABD system (4.18) if we rearrange it as follows

$$
\widetilde{A} = \begin{pmatrix}
-I & I & & & & & & & & & \\
S_0 & 0 & T_1 & R_1 & & & & & & & \\
 & -I & 0 & 0 & I & & & & & & \\
 & & & S_1 & 0 & T_2 & R_2 & & & & \\
 & & & & -I & 0 & 0 & I & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & S_{N-1} & 0 & T_N & R_N & \\
 & & & & & & & -I & 0 & 0 & I \\
 & & & & & & & & & D_b & D_a
\end{pmatrix},
$$

$$
\widetilde{x} = \begin{pmatrix} \mathbf{z}_0 & \mathbf{y}_0 & \mathbf{w}_1 & \mathbf{z}_1 & \mathbf{y}_1 & \mathbf{w}_2 & \dots & \mathbf{z}_{N-1} & \mathbf{y}_{N-1} & \mathbf{w}_N & \mathbf{z}_N & \mathbf{y}_N \end{pmatrix},
$$

$$
\widetilde{f} = \begin{pmatrix} 0 & \mathbf{d} & 0 & \mathbf{f}_1 & 0 & \mathbf{f}_2 & \dots & 0 & \mathbf{f}_N \end{pmatrix},
\tag{4.33}
$$

where S_i, T_i, R_i, $\mathbf{d}$, $\mathbf{f}_i$, $\mathbf{z}_i$, $\mathbf{w}_i$ are the same blocks and vectors as in (4.18) and vectors $\mathbf{y}_i$ are new unknowns:

$$
\mathbf{y}_N = \ldots = \mathbf{y}_1 = \mathbf{y}_0 = \mathbf{z}_0.
$$

Then, we can apply *COLROW* to the ABD linear system (4.33) and compare its results to those of *GBABDCR* and *GBABDCR-0* applied to the original BABD linear system (4.18). Now, the cost of *COLROW* in the factorization phase is

$$
\begin{array}{l}
N(\frac{2}{3}\tilde{k}^3 + 5\tilde{k}^2 m + 15\tilde{k}m^2 + \frac{46}{3}m^3 - \frac{1}{2}\tilde{k}^2 - 4\tilde{k}m - 5m^2 - \frac{1}{6}\tilde{k} - \frac{1}{3}m) \\
+3\tilde{k}m^2 + \frac{25}{3}m^3 - \tilde{k}m - 3m^2 - \frac{1}{3}m.
\end{array}
\tag{4.34}
$$

From (4.34) and (4.31), we note that *GBABDCR* saves with respect *COLROW*

$$
\begin{array}{l}
N\left(3\tilde{k}^2 m + +11\tilde{k}m^2 + \frac{32}{3}m^3 - 3\tilde{k}m - \frac{7}{2}m^2 - \frac{1}{6}m\right) \\
+3\tilde{k}m^2 + \frac{25}{3}m^3 - \tilde{k}m - 3m^2 - \frac{1}{3}m
\end{array}
$$

flops.

For each test case shown in Tables 4.1-4.2, the coefficients of the BABD coefficient matrix A have been generated randomly. We use a AlphaServer DS20E with a 667 MHz EV67 processor and a Compaq Fortran 90 (formerly Digital Fortran 90) compiler. The right hand side $\mathbf{f}$ has been set so that the solution is $(1, \ldots, 1)^T$.

In Table 4.1, we show the execution times, in seconds, as functions of $\tilde{k}$, m and N, varying each parameter in turn and fixing the others. The timings confirm

Table 4.1: Execution times (in seconds) to solve linear systems with varying m, $\tilde{k}$ and N using *GBABDCR*, *GBABDCR-0* and *COLROW*.

problem	*GBABDCR*	*GBABDCR-0*	*COLROW*
$m = 5, \tilde{k} = 10, N = 2000$	6.53e-02	9.86e-02	0.148
$m = 10, \tilde{k} = 10, N = 2000$	0.156	0.222	0.439
$m = 20, \tilde{k} = 10, N = 2000$	0.529	0.651	1.581
$m = 10, \tilde{k} = 5, N = 2000$	0.106	0.135	0.317
$m = 10, \tilde{k} = 10, N = 2000$	0.156	0.222	0.439
$m = 10, \tilde{k} = 20, N = 2000$	0.305	0.490	0.733
$m = 10, \tilde{k} = 10, N = 1000$	7.12e-02	0.105	0.208
$m = 10, \tilde{k} = 10, N = 2000$	0.156	0.222	0.439
$m = 10, \tilde{k} = 10, N = 4000$	0.345	0.463	0.876

Table 4.2: Computed errors in the solution of linear systems with varying m, $\tilde{k}$ and N using *GBABDCR*, *GBABDCR-0* and *COLROW*.

problem	*GBABDCR*	*GBABDCR-0*	*COLROW*
$m = 5, \tilde{k} = 10, N = 2000$	4.12e-07	4.09e-07	9.95e-07
$m = 10, \tilde{k} = 10, N = 2000$	5.56e-10	9.75e-10	3.53e-10
$m = 20, \tilde{k} = 10, N = 2000$	1.32e-10	6.40e-11	8.15e-11
$m = 10, \tilde{k} = 5, N = 2000$	4.72e-11	7.88e-11	3.47e-11
$m = 10, \tilde{k} = 10, N = 2000$	5.56e-10	9.75e-10	3.53e-10
$m = 10, \tilde{k} = 20, N = 2000$	4.13e-11	3.41e-11	3.71e-11
$m = 10, \tilde{k} = 10, N = 1000$	7.08e-11	4.04e-11	4.61e-11
$m = 10, \tilde{k} = 10, N = 2000$	5.56e-10	9.75e-10	3.53e-10
$m = 10, \tilde{k} = 10, N = 4000$	5.56e-10	1.09e-09	3.53e-10

our expectations from theory: *GBABDCR* is faster than *GBABDCR-0* and *COLROW*, though the speed-ups are lower than theory suggests. Moreover, the gap between *GBABDCR* and *GBABDCR-0* increases for larger values of $\tilde{k}/m$; conversely, *GBABDCR* improves over *COLROW* when $m/\tilde{k}$ increases. The ratios of costs between *GBABDCR-0* and *GBABDCR* and between *COLROW* and *GBABDCR* are approximately 1.5 and 2.8, respectively. In Table 4.2, we show the errors of the methods in the same tests used in Table 4.1. Note that the order of the error is the same for all the methods. The case $m = 5, \tilde{k} = 10, N = 2000$ gives a slight larger than expected error, but this is true for all the methods and probably depends on the chosen random matrix.

4.6 An approach to solve BABD systems in the monomial case

We consider the solution of BABD linear systems with structure

$$\begin{pmatrix} D_a & & & & & & D_b \\ H_1 & G_1 & & & & & \\ -E_1 & -F_1 & I & & & & \\ & & \ddots & & & & \\ & & & & H_N & G_N & \\ & & & & -E_N & -F_N & I \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{w}_1 \\ \mathbf{z}_2 \\ \vdots \\ \vdots \\ \mathbf{z}_N \\ \mathbf{w}_N \\ \mathbf{z}_{N+1} \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ 0 \\ \vdots \\ \mathbf{f}_N \\ 0 \end{pmatrix}, \tag{4.35}$$

where $-E_i \in \mathbb{R}^{m\times m}$, the identity matrix $I \in \mathbb{R}^{m\times m}$, $H_i \in \mathbb{R}^{\tilde{k}\times m}$, $-F_i \in \mathbb{R}^{m\times \tilde{k}}$ and $G_i \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$. Those systems arise from OSC with monomial basis functions, see Section 1.7.2.

4.6.1 The algorithm *BABDCR_MONO*

We describe a particular cyclic reduction approach for solving the BABD system (4.35), called *BABDCR_MONO*. This algorithm takes advantage of the structure in (4.35) through factorizations of matrices of smaller dimension with respect to that used in the cases in Figure 4.1, proposed in Section 4.2. Here the blocks S_{i-1}, T_i and R_i are those defined in (4.8).

The algorithm transforms each pair of block row equations

$$\begin{pmatrix} H_i & G_i & & & \\ -E_i & -F_i & I & & \\ & & H_{i+1} & G_{i+1} & \\ & & -E_{i+1} & -F_{i+1} & I \end{pmatrix} \begin{pmatrix} \mathbf{z}_i \\ \mathbf{w}_i \\ \mathbf{z}_{i+1} \\ \mathbf{w}_{i+1} \\ \mathbf{z}_{i+2} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i \\ 0 \\ \mathbf{f}_{i+1} \\ 0 \end{pmatrix}, \tag{4.36}$$

for $i = 2j-1,\ j = 1, 2, \ldots, \lfloor N/2 \rfloor$, into one block row equation

$$\begin{pmatrix} H_i' & G_i' & \\ -E_i' & -F_i' & J_{i+2}' \end{pmatrix} \begin{pmatrix} \mathbf{z}_i \\ \mathbf{w}_i \\ \mathbf{z}_{i+2} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i' \\ \mathbf{g}_i' \end{pmatrix}, \tag{4.37}$$

involving only the unknowns $\mathbf{z}_i$, $\mathbf{w}_i$ and $\mathbf{z}_{i+2}$. Blocks H_i', G_i' and J_{i+2}' have respectively the same number of columns as H_i, G_i and I. The blocks H_i' (and $-E_i'$) overlap the columns of J_i', that are generated by the reduction of the previous pair of block rows. Also, J_{i+2}' overlaps the columns of H_{i+2}' (and $-E_{i+2}'$), that are generated by the reduction of the successive pair of block rows. To do this, we

use a cyclic reduction approach similar to that presented in Section 4.2 for M_i in case 2, Figure 4.1. Since the coefficient matrix (4.35) is non-singular, the columns overlapped by the $(\tilde{k}+2m)\times(\tilde{k}+m)$ matrix M_i

$$M_i = \begin{pmatrix} I & \\ H_i & G_i \\ -E_i & -F_i \end{pmatrix}$$

are linear independent. Hence, as previously, we use a row partial pivoting LU factorization

$$P_i M_i = \begin{pmatrix} L_i \\ C_i \end{pmatrix} U_i = \begin{pmatrix} I & \\ C_i L_i^{-1} & I \end{pmatrix} \begin{pmatrix} L_i U_i \\ O \end{pmatrix}, \tag{4.38}$$

where P_i is a $(\tilde{k}+2m)\times(\tilde{k}+2m)$ permutation matrix, L_i and U_i are lower and upper triangular matrices of dimension $(\tilde{k}+m)\times(\tilde{k}+m)$, respectively. We premultiply (4.36) by $\begin{pmatrix} I & \\ & P_i \end{pmatrix}$ and the inverse of the lower triangular matrix $\begin{pmatrix} I & & \\ & I & \\ & C_i L_i^{-1} & I \end{pmatrix}$, yielding

$$\begin{pmatrix} H_i' & G_i' & & \\ \tilde{E}_i & \tilde{F}_i & L_i U_i & \tilde{J_{i+2}} \\ -E_i' & -F_i' & & J_{i+2}' \end{pmatrix} = \begin{pmatrix} I & & \\ & I & \\ & -C_i L_i^{-1} & I \end{pmatrix} \begin{pmatrix} I & \\ & P_i \end{pmatrix} \begin{pmatrix} H_i & G_i & & \\ -E_i & -F_i & I & \\ & & H_{i+1} & G_{i+1} \\ & & -E_{i+1} & -F_{i+1} & I \end{pmatrix}. \tag{4.39}$$

The corresponding right hand sides are reduced, using the same operations:

$$\begin{pmatrix} \mathbf{f}_i' \\ \tilde{\mathbf{f}}_i \\ \mathbf{g}_i' \end{pmatrix} = \begin{pmatrix} I & & \\ & I & \\ & -C_i L_i^{-1} & I \end{pmatrix} \begin{pmatrix} I & \\ & P_i \end{pmatrix} \begin{pmatrix} \mathbf{f}_i \\ 0 \\ \mathbf{f}_{i+1} \\ 0 \end{pmatrix}. \tag{4.40}$$

The blocks in (4.39) have the following sizes: $H_i' \in \mathbb{R}^{\tilde{k}\times m}$, $G_i' \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$, $-E_i' \in \mathbb{R}^{m\times m}$, $-F_i' \in \mathbb{R}^{m\times\tilde{k}}$, $J_{i+2}' \in \mathbb{R}^{m\times m}$, and $\tilde{E}_i \in \mathbb{R}^{(\tilde{k}+m)\times\tilde{k}}$, $\tilde{F}_i \in \mathbb{R}^{(\tilde{k}+m)\times\tilde{k}}$, $\tilde{J_{i+2}} \in \mathbb{R}^{(\tilde{k}+m)\times m}$. The vectors in (4.40) have length: $\mathbf{f}_i' \in \mathbb{R}^{\tilde{k}}$, $\mathbf{g}_i' \in \mathbb{R}^m$ and $\tilde{\mathbf{f}}_i \in \mathbb{R}^{\tilde{k}+m}$. From (4.39) and (4.40), it is easy to prove that the blocks into the reduced row block equation (4.37) are determined through the operations

$$\begin{aligned} H_i' = H_i, \qquad G_i' = G_i, \\ \begin{pmatrix} -E_i' & -F_i' \end{pmatrix} &= \begin{pmatrix} -C_i L_i^{-1} & I \end{pmatrix} P_i \begin{pmatrix} -E_i & -F_i \\ & O \end{pmatrix}, \\ J_{i+2}' &= \begin{pmatrix} -C_i L_i^{-1} & I \end{pmatrix} P_i \begin{pmatrix} O \\ I \end{pmatrix}, \end{aligned}$$

and

$$\mathbf{f}'_i = \mathbf{f}_i, \qquad \mathbf{g}'_i = \begin{pmatrix} -C_i L_i^{-1} & I \end{pmatrix} P_i \begin{pmatrix} 0 \\ \mathbf{f}_i \\ 0 \end{pmatrix}.$$

Note that J'_{i+2} is not an identity matrix.
After l steps of reduction, $l = 1, \ldots, \log_2(N) - 2$, the coefficient matrix obtained is of size $\lceil N/s \rceil (\tilde{k} + m) + m$ where $s = 2^l$, and it can be further on reduced by combining two adjacent block row equations (for each $i = 2(j-1)s + 1$, with $j = 1, 2, \ldots, \lfloor N/(2s) \rfloor$)

$$\begin{pmatrix} H_i^{(l)} & G_i^{(l)} & & & \\ -E_i^{(l)} & -F_i^{(l)} & J_{i+s}^{(l)} & & \\ & & H_{i+s}^{(l)} & G_{i+s}^{(l)} & \\ & & -E_{i+s}^{(l)} & -F_{i+s}^{(l)} & J_{i+2s}^{(l)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_i \\ \mathbf{w}_i \\ \mathbf{z}_{i+s} \\ \mathbf{w}_{i+s} \\ \mathbf{z}_{i+2s} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i^{(l)} \\ \mathbf{g}_i^{(l)} \\ \mathbf{f}_{i+s}^{(l)} \\ \mathbf{g}_{i+s}^{(l)} \end{pmatrix},$$

in order to obtain

$$\begin{pmatrix} H_i^{(l+1)} & G_i^{(l+1)} & \\ -E_i^{(l+1)} & -F_i^{(l+1)} & J_{i+2s}^{(l+1)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_i \\ \mathbf{w}_i \\ \mathbf{z}_{i+2s} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i^{(l+1)} \\ \mathbf{g}_i^{(l+1)} \end{pmatrix}.$$

Let $p = \lceil \log_2(N) \rceil$, after $p - 1$ steps the linear system has the two block row equations

$$\begin{pmatrix} H_1^{(p-1)} & G_1^{(p-1)} & & & & \\ -E_1^{(p-1)} & -F_1^{(p-1)} & J_{\frac{N}{2}+1}^{(p-1)} & & & \\ & & H_{\frac{N}{2}+1}^{(p-1)} & G_{\frac{N}{2}+1}^{(p-1)} & & \\ & & -E_{\frac{N}{2}+1}^{(p-1)} & -F_{\frac{N}{2}+1}^{(p-1)} & J_{N+1}^{(p-1)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{w}_1 \\ \mathbf{z}_{\frac{N}{2}+1} \\ \mathbf{w}_{\frac{N}{2}+1} \\ \mathbf{z}_{N+1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1^{(p-1)} \\ \mathbf{g}_1^{(p-1)} \\ \mathbf{f}_{\frac{N}{2}+1}^{(p-1)} \\ \mathbf{g}_{\frac{N}{2}+1}^{(p-1)} \end{pmatrix}, \tag{4.41}$$

it can be further on reduced in order to obtain a system with a full coefficient matrix of size $2m \times 2m$

$$\begin{pmatrix} D_a & D_b \\ H_1^{(p)} & J_{N+1}^{(p)} \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_{N+1} \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1^{(p)} \end{pmatrix}. \tag{4.42}$$

To do this, we use a row partial pivoting LU factorization

$$P_1^{(p-1)} \begin{pmatrix} G_1^{(p-1)} & & \\ -F_1^{(p-1)} & J_{\frac{N}{2}+1}^{(p-1)} & \\ & H_{\frac{N}{2}+1}^{(p-1)} & G_{\frac{N}{2}+1}^{(p-1)} \\ & -E_{\frac{N}{2}+1}^{(p-1)} & -F_{\frac{N}{2}+1}^{(p-1)} \end{pmatrix} = \begin{pmatrix} I & \\ C_1^{(p-1)}(L_1^{(p-1)})^{-1} & I \end{pmatrix} \begin{pmatrix} L_1^{(p-1)}U_1^{(p-1)} \\ 0 \end{pmatrix},$$

where $P_1^{(p-1)}$ is a $2(\tilde{k}+m) \times 2(\tilde{k}+m)$ permutation matrix, $L_1^{(p-1)}$ and $U_1^{(p-1)}$ are lower and upper triangular matrices of dimension $(2\tilde{k}+m) \times (2\tilde{k}+m)$, and $C_1^{(p-1)}$ is an $m \times (2\tilde{k}+m)$ matrix. Afterwards, we premultiply (4.41) by $P_1^{(p-1)}$ and by $\begin{pmatrix} I & \\ C_1^{(p-1)}(L_1^{(p-1)})^{-1} & I \end{pmatrix}^{-1}$. These operations yield

$$\begin{pmatrix} \tilde{H}_1 & L_1^{(p-1)}U_1^{(p-1)} & \tilde{J}_1 \\ H_1^{(p)} & & J_{N+1}^{(p)} \end{pmatrix} = \begin{pmatrix} I & \\ -C_1^{(p-1)}(L_1^{(p-1)})^{-1} & I \end{pmatrix} P_1^{(p-1)} \cdot \begin{pmatrix} H_1^{(p-1)} & G_1^{(p-1)} & & \\ -E_{\frac{N}{2}}^{(p-1)} & -F_{\frac{N}{2}}^{(p-1)} & J_{\frac{N}{2}}^{(p-1)} & \\ & & H_{\frac{N}{2}+1}^{(p-1)} & G_{\frac{N}{2}+1}^{(p-1)} \\ & & -E_{\frac{N}{2}+1}^{(p-1)} & -F_{\frac{N}{2}+1}^{(p-1)} & J_{N+1}^{(p-1)} \end{pmatrix},$$

$$\begin{pmatrix} \tilde{\mathbf{f}}_1 \\ \mathbf{f}_1^{(p)} \end{pmatrix} = \begin{pmatrix} I & \\ C_1^{(p-1)}(L_1^{(p-1)})^{-1} & I \end{pmatrix} P_1^{(p-1)} \begin{pmatrix} \mathbf{f}_1^{(p-1)} \\ \mathbf{g}_1^{(p-1)} \\ \mathbf{f}_{\frac{N}{2}+1}^{(p-1)} \\ \mathbf{g}_{\frac{N}{2}+1}^{(p-1)} \end{pmatrix},$$

where $\tilde{H}_1, \tilde{J}_1 \in \mathbb{R}^{(2\tilde{k}+m)\times m}$, $\tilde{\mathbf{f}}_1 \in \mathbb{R}^{2\tilde{k}+m}$ and $H_1^{(p)}$, $J_{N+1}^{(p)} \in \mathbb{R}^{m\times m}$, $\mathbf{f}_1^{(p)} \in \mathbb{R}^{2\tilde{k}+m}$ are the blocks and right hand side into the $2m$ by $2m$ system (4.42).

The algorithm proceeds with the solution of (4.42) and the back-substitution phase where all the other unknowns $\mathbf{w}_1, \mathbf{z}_2, \mathbf{w}_2 \ldots, \mathbf{z}_N, \mathbf{w}_N$ are computed.

4.6.2 Description of the software

The *BABDCR_ MONO* package has two main subroutines:

- BABDCR_ MONO_FACT performs the factorization of the coefficient matrix of the system (4.35);
- BABDCR_ MONO_SOLV uses the output of BABDCR_ MONO_FACT for solving (4.35).

In figure (4.4) the calling sequences of the two subroutines are shown.

SUBROUTINE BABDCR_MONO_FACT($\tilde{k}$, m, N, ARRAY1, ARRAY2, LFTBLK, RGTBLK, PERM, FILL_IN, INFO)

SUBROUTINE BABDCR_MONO_SOLV($\tilde{k}$, m, N, ARRAY1, ARRAY2, LFTBLK, RGTBLK, PERM, FILL_IN, VECT_B)

Figure 4.4: Calling sequences of the two main subroutines

The package requires that the coefficient matrix in input is given as in Figure 4.5, that is, blocks (H_i, G_i, O) and $(-E_i, -F_i, I)$ must be given sequentially and saved respectively in ARRAY1 and ARRAY2 which are three-dimensional arrays of size $\tilde{k} \times (\tilde{k} + 2m) \times N$ and $m \times (\tilde{k} + 2m) \times N$.
The right hand side of (4.35) must be given in a vector VECT_B of length $(\tilde{k} + m)N + m$, see Figure 4.6.2. The boundary blocks D_a and D_b are stored, in the arrays LFTBLK and RGTBLK, respectively, both of size $m \times m$.

$$\text{ARRAY1} = \left(\begin{array}{|c|c|c|} \hline H_1 & G_1 & O \\ \hline \end{array} \ \ldots\ldots \ \begin{array}{|c|c|c|} \hline H_N & G_N & O \\ \hline \end{array} \right)$$

$$\text{ARRAY2} = \left(\begin{array}{|c|c|c|} \hline -E_1 & -F_1 & I \\ \hline \end{array} \ \ldots\ldots \ \begin{array}{|c|c|c|} \hline -E_N & -F_N & I \\ \hline \end{array} \right)$$

Figure 4.5: Structure of the input coefficient matrix

$$\text{VECT\_B} = [\mathbf{d}, \mathbf{f}_1, 0, \mathbf{f}_2, 0, \ldots, \mathbf{f}_N, 0]$$

Figure 4.6: Structure of the right hand side

4.7 Comparisons in the monomial case

We compare the code *ABDPACK*, described in Section 2.5.1, with our algorithms: *GBABDCR* and *BABDCR_MONO*. *ABDPACK* solves ABD linear systems with coefficient matrices having the structure in (2.14). Our tests have been employed on BABD systems with structure (4.35) and on the same machine used in the numerical test in Section 4.5. We apply *ABDPACK* to the equivalent large ABD linear system, obtained through a transformation of the initial BABD system (4.35). This transformation is similar to that shown in (4.33) where the blocks S_{i-1}, T_i and R_i are those represented in (4.8). We compare its results with respect to *GBABDCR* and *BABDCR_MONO* applied to the initial system (4.35). We have tested these methods on randomly generated BABD systems with $\tilde{k} = 6$, $m = 4$ and $N = 256, 512, 1024, 2048, 4096$.
From Table 4.4, we note that the cyclic reduction method *GBABDCR* is at least 10% faster than *BABDCR_MONO*. Since *ABDPACK* is applied to an ABD system, we have inserted in Table 4.3 the timings for obtaining the double sized ABD system from the original BABD system. We note from Tables 4.4 and 4.3, that the code *ABDPACK* is slower than the two cyclic reduction algorithms just for the transformation of the original system. Errors of the three linear solvers, represented in Table 4.5, depend on the conditioning of the system. This topic needs more testing and numerical analysis to find how the stability of the codes models the accuracy of the results. We obtain that, for N large ($N > 1024$), *BABDCR_MONO* has an error of order 10^{-7} , while, the other two methods are slightly more accurate (the error is of order 10^{-8}). For $N < 1024$, the more accurate methods are *BABDCR_MONO* and *ABDPACK*. Hence, it is preferable to use *GBABDCR* for a large number of block rows N and *BABDCR_MONO* for small N. This first case is clearly of interest in most of the BVPs.

If more than one system is solved, it can be convenient to use *BABDCR_MONO* just because the time spent in the solution phase is less than in *GBABDCR*, as shown in Table 4.6.

Concerning memory allocation, we note that *BABDCR_MONO* requires the same storage size of *GBABDCR*. *ABDPACK* requires less storage because it generates no fill-in and it not requires the null block of ARRAY1 as *GBABDCR* and *BABDCR_MONO* do, see Figure 4.5.

Table 4.3: Timings of assemble-phase (converting a BABD in an ABD system) for the method *ABDPACK*, with $\tilde{k} = 6$, $m = 4$ and $N = 256, 512, \ldots, 4096$

	N=256	N=512	N=1024	N=2048	N=4096
ABDPACK	1.95e-03	4.88e-03	8.78e-03	1.95e-02	4.68e-02

Table 4.4: Total timings for the methods *BABDCR_MONO*, *GBABDCR*, *ABDPACK*, with $\tilde{k} = 6$, $m = 4$ and $N = 256, 512, \ldots, 4096$ applied to random BABD linear systems

	N=256	N=512	N=1024	N=2048	N=4096
BABDCR_MONO	4.88e-03	1.07e-02	2.15e-02	4.78e-02	8.59e-02
GBABDCR	4.88e-03	9.76e-03	1.95e-02	4.39e-02	7.71e-02
ABDPACK	7.81e-03	1.66e-02	3.22e-02	6.44e-02	0.15

Table 4.5: Errors for the methods *BABDCR_MONO*, *GBABDCR*, *ABDPACK*, with $\tilde{k} = 6$, $m = 4$ and $N = 256, 512, \ldots, 4096$ applied to random BABD linear systems

	N=256	N=512	N=1024	N=2048	N=4096
BABDCR_MONO	5.15e-10	4.93e-12	1.46e-09	1.35e-07	1.09e-07
GBABDCR	1.42e-09	1.56e-11	3.83e-09	3.47e-08	3.31e-08
ABDPACK	2.54e-10	1.22e-11	4.30e-10	3.90e-08	6.52e-08

Table 4.6: Timings of solution-phase for the methods *BABDCR_MONO*, *GBABDCR*, *ABDPACK*, with $\tilde{k} = 6$, $m = 4$ and $N = 256, 512, \ldots, 4096$ applied to random BABD linear systems

	N=256	N=512	N=1024	N=2048	N=4096
BABDCR_MONO	9.76e-04	2.93e-03	5.86e-03	1.46e-02	2.34e-02
GBABDCR	1.95e-03	2.93e-03	6.83e-03	1.46e-02	2.64e-02
ABDPACK	9.76e-04	2.93e-03	4.88e-03	1.07e-02	2.73e-02

Chapter 5

A parallel algorithm for the solution of BABD systems using MPI

We analyze an MPI Fortran 90 package, called *PBABDCR*, for the solution of BABD linear systems with the structure

$$\begin{pmatrix} D_a & & & & D_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix}. \tag{5.1}$$

The package handles BABD matrices with square blocks S_i, R_i, D_a and D_b all of size $m \times m$.

5.1 The parallel algorithm

PBABDCR is a parallel implementation on P processors of the cyclic reduction algorithm *BABDCR*, described in Chapter 3. The package requires that the coefficient matrix and the right hand side of the system (5.1) are defined as in Figure 5.1: the blocks rows $V_i = (S_{i-1}, R_i)$ and the boundary blocks D_a and D_b must be given sequentially in an $m \times m \times 2(N+1)$ real array MATR_A and the right hand side in an $m \times (N+1)$ real array VECT_B.

In the following, we indicate a processor using directly its handle K, where $K = 0, \ldots, P-1$.

The algorithm consists in three phases:

1. STARTUP
2. FACTORIZATION

$$\text{MATR\_A} = \left(\boxed{V_1} \boxed{V_2} \ldots \boxed{V_N} \boxed{D_a} \boxed{D_b} \right),$$
$$\text{VECT\_B} = \left(\boxed{\mathbf{d}} \boxed{\mathbf{f}_1} \boxed{\mathbf{f}_2} \ldots \boxed{\mathbf{f}_N} \right)$$

Figure 5.1: Structures of the input coefficient matrix MATR_A and of the input right hand side VECT_B.

3. SOLUTION

In the STARTUP phase, proc 0 breaks the blocks

$$\left(\boxed{V_1} \boxed{V_2} \ldots \boxed{V_N} \right), \left(\boxed{\mathbf{f}_1} \boxed{\mathbf{f}_2} \ldots \boxed{\mathbf{f}_N} \right),$$

into contiguous parts and distributes them to all the P processors.

As an example, after this block-distribution, proc[1] K has $N_0 = \lfloor N/P \rfloor$ block rows:

$$V_{N_0K+1}, \ldots, V_{N_0(K+1)},$$

and the associated right hand sides

$$\mathbf{f}_{N_0K+1}, \ldots, \mathbf{f}_{N_0(K+1)}.$$

If N is not a multiple of K, proc $P-1$ also has the last $N - N_0P$ block rows:

$$V_{N_0P+1}, \ldots, V_N,$$

and the associated right hand sides

$$\mathbf{f}_{N_0P+1}, \ldots, \mathbf{f}_N.$$

Moreover, proc 0 broadcasts to each processor the boundary blocks D_a, D_b and the associated right hand side $\mathbf{d}$.

Property 5.1.1. The algorithm requires that each processor K, with $K = 0, \ldots, P-2$, has number of block rows a power of 2, that is $N_0 = 2^i$, for some $i \in \mathbb{N}$.

Property 5.1.1 could represent a drawback of the parallel code, if the choice of the number of processors is user-defined.

For a given parallel architecture and problem size, the algorithm, in the startup phase, chooses the number of processors to minimize the number of steps in the factorization and solution phases. This part is described in detail in Section 5.1.3.

[1] We use the term 'proc' as an abbreviation for 'processor'

5.1.1 Factorization phase

The factorization phase consists in $\log_2 N$ steps which perform the cyclic reduction, described in Section 3.1, for the algorithm *BABDCR*. One difference with respect to *BABDCR* is that the computation in each step is clearly distributed between processors and we need to communicate between them to proceed with the reduction.

Now, set $(K, j) = N_0 K + j$, for $j = 1, \ldots, N_0$, we start the factorization supposing that proc K has stored the matrix

$$\begin{pmatrix} S_{(K,0)} & R_{(K,1)} & & \\ & \ddots & & \\ & & S_{(K,N_0-1)} & R_{(K,N_0)} \end{pmatrix} \tag{5.2}$$

in its own memory allocations, as described in the previous section.

The $\log_2 N$ steps of the factorization are grouped in two parts:

- In the first $k_0 = \lceil \log_2(N_0) \rceil$ steps, called the reduction steps, the coefficient matrix (5.2) is reduced into the block row

$$\begin{pmatrix} S^{(k_0)}_{(K,0)} & R^{(k_0)}_{(K,N_0)} \end{pmatrix}, \tag{5.3}$$

  without any communication between processors. This yields, altogether in the parallel architecture, the BABD coefficient matrix

$$\begin{pmatrix} D_a & & & D_b \\ S^{(k_0)}_{(0,0)} & R^{(k_0)}_{(0,N_0)} & & \\ & \ddots & & \\ & & S^{(k_0)}_{(P-1,0)} & R^{(k_0)}_{(P-1,N_0)} \end{pmatrix}.$$

- in the next step (called the communication step), each processor K receives a block row from an adjacent processor (that has handle $K-1$, if K is odd, and $K+1$, if K is even). Then, the processors K and $K+1$ (with $K = 2j$, for $j = 0, \ldots, \lfloor \frac{P-2}{2} \rfloor$) have the same pair of block rows[2]

$$\begin{pmatrix} S^{(k_0)}_{(K,0)} & R^{(k_0)}_{(K,N_0)} & \\ & S^{(k_0)}_{(K+1,0)} & R^{(k_0)}_{(K+1,N_0)} \end{pmatrix},$$

  that is reduced into the block row

$$\begin{pmatrix} S^{(k_0+1)}_{(K,0)} & R^{(k_0+1)}_{(K+1,N_0)} \end{pmatrix}. \tag{5.4}$$

  In the next step (a communication step), the pairs of processors $K, K+2$ and $K+1, K+3$ for $K = 4j$, $j = 0, 1, \ldots$, interchange their last determined block rows and reduce the obtained pair of block rows. This algorithm

[2]Note that $(K, N_0) = (K+1, 0)$

proceeds for $lnp = \lceil \log_2(P) \rceil$ steps until each processor factorizes the final 2×2 block coefficient matrix

$$\begin{pmatrix} D_a & D_b \\ S_0^{(k_0+lnp)} & R_N^{(k_0+lnp)} \end{pmatrix}. \tag{5.5}$$

For more details, see the algorithm listed in Section 5.2.

5.1.2 Solution phase

The SOLUTION phase consists in three parts

- in the first k_0 steps (reduction steps), the right hand side is reduced without any communication between processors. This produces the right hand sides associated to the coefficient matrices determined in the first k_0 steps of the factorization phase;
- in the next $\lceil \log_2(P) \rceil$ steps (communication steps), the right hand side is reduced after communication between processors (those used in the second part of the factorization phase);
- finally, the back-substitution phase is performed in each processor. It determines the unknowns of the system (5.1), without any additional communication between processors.

5.1.3 A note about the startup

First, we analyze how to choose the number of processors for minimizing the computational time spent in the cyclic reduction. In this analysis, we consider the time spent for the communications between processors to be negligible.

Given N block rows and a parallel architecture with $MAXP$ processors, the number of processors P and the number of block rows N_0, that minimize the computational time of the reduction, are determined from the following

Proposition 5.1.2. *If $\lfloor \frac{N}{2} \rfloor = l$ with $l \leq MAXP$, then $P = l$ and $N_0 = 2$, in the first $P-1$ processors, and $N_0 = N - 2(P-1)$ in proc $P-1$. Otherwise, if $l > MAXP$, the number of processors, P, and the number of block rows, N_0, is determined from the following algorithm*

$l = \lfloor \frac{N}{2} \rfloor$
$n = 0$
`repeat`
  $l = \lceil \frac{l}{2} \rceil$
  $n = n + 1$
`until` $(l \leq MAXP)$
$P = l$
$N_0 = 2^{n+1}$ `for any proc with handle` $0, \ldots, P-2$

$N_0 = N - (P-1)2^{n+1}$ `for proc` $P-1$.

Proposition 5.1.3. *Given N block rows, a parallel architecture with $MAXP$ processors, and using the approach described in the previous proposition, we need k_0 factorization steps (before the communication phase), with $N_0 = 2^{k_0}$ for proc K with $K = 0, \ldots, P-2$ and $N_0 \leq 2^{k_0} + 1$ for proc $P-1$. Therefore, the Proposition 5.1.2 yields an algorithm that satisfies the Property 5.1.1.*

Below, we refer to a time step as a single reduction (from a pair of block rows to one block row), as described in Section 3.1. Also, we refer to timing as the time spent in a time step.

Proposition 5.1.4. *The values P and N_0 determined in the Proposition 5.1.2, yields an algorithm that solves* (5.1) *with the minimum possible number of time steps, for a given parallel architecture.*

We illustrate the Proposition 5.1.4, using the following two examples. In the first, we use a parallel architecture with $MAXP = 3$ processors and a BABD system (5.1) with $N = 5$ block rows. If we use all the three processors, in the algorithm, then $P = 3$ and in order to satisfy the property 5.1.1, V_1 is stored in proc 0, V_2 in proc 1 and V_3, V_4, and V_5 in proc 2. This results in an algorithm where only proc 2 performs the reduction of its blocks (two reductions), while the other processors wait for the communication phase. The communication phase consists of 2 communication steps and therefore 2 additional reductions. Thus in total (in the parallel machine), we use 4 time steps with 2 communications for obtaining the final 2×2 block linear system with coefficient matrix (5.5) in each processor. In contrast, if we use, as recommended by Proposition 5.1.2, only two processors, then $P = 2$ with V_1 and V_2 in proc 0 and V_3, V_4, V_5 in proc 1. This yields one reduction performed in both processors, 2 communication steps and the associated reductions. Then, we use only 3 time steps and 2 communications.

An other example: we set $MAXP = 2$ and $N = 15$. Since $l = \lfloor \frac{15}{3} \rfloor > MAXP$, from the algorithm in Proposition 5.1.2 we get $P = \lceil \lceil \frac{l}{2} \rceil / 2 \rceil = 2$, $N_0 = 2^3 = 8$ for proc 0 and $N_0 = 7$ for proc 1. This yields, before the communication, 7 time steps (with $k_0 = 3$) and after this, there is one communication step with the associated reduction. We use a total of 8 time-steps and 1 communication.

In contrast, if we use only one processor this yields $N_0 = 15$ then there are $k_0 = 4$ reduction steps. These yield 7 time steps in the first reduction step, 4 in the second step, 2 in the third step and 1 time step in the fourth reduction step. Therefore we get 14 time steps.

5.2 Code in a pseudo programming language

The algorithm, presented below in a pseudo-programming language, describes the factorization and the solution phase in the case: $P = 2^{lnp}$ and $N_0 = 2^{k_0}$.

FACTORIZATION in proc K

$k_0 = \lceil log_2(N_0) \rceil$
$lnp = \lceil log_2(P) \rceil$
% k_0 reduction steps
for $step = 0, \ldots, k_0 - 1$
  $s = 2^{step}$
  for $j = 1, 2, \ldots, \lfloor \lceil N_0/s \rceil / 2 \rfloor$
    % reduction of a pair of block rows
    $l = (K, (2j-1)s)$

$$P_l \begin{pmatrix} R_l^{(step)} \\ S_l^{(step)} \end{pmatrix} = \begin{pmatrix} L_l \\ T_l \end{pmatrix} U_l$$

$$G_l^{(step)} = T_l L_l^{-1}$$

$$\begin{pmatrix} \widehat{S}_{l-s}^{(step)} \\ S_{l-s}^{(step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(step)} & I \end{pmatrix} P_l \begin{pmatrix} S_{l-s}^{(step)} \\ O \end{pmatrix}$$

$$\begin{pmatrix} \widehat{R}_{l+s}^{(step)} \\ R_{l+s}^{(step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(step)} & I \end{pmatrix} P_l \begin{pmatrix} O \\ R_{l+s}^{(step)} \end{pmatrix}$$

  end
end
$s_1 = 2$
$prod = mod(K, s_1)$
% lnp communication steps
  $l_1(0) = (K, 0)$
  $l_2(0) = (K, N_0)$
  for $step = 0, \ldots, lnp - 1$
    IF $(prod == 0)$ THEN
    % communication with $K + \frac{s_1}{2}$
      send $\begin{pmatrix} S_{l_1(step)}^{(k_0+step)} & R_{l_2(step)}^{(k_0+step)} \end{pmatrix}$ to proc $K + \frac{s_1}{2}$
      $m_1(step) = (k + \frac{s_1}{2}, 0)$
      $m_2(step) = (k + \frac{s_1}{2}, N_0)$
      receive $\begin{pmatrix} S_{m_1(step)}^{(k_0+step)} & R_{m_2(step)}^{(k_0+step)} \end{pmatrix}$ from proc $K + \frac{s_1}{2}$
      $l_1(step+1) = l_1(step)$
      $l_2(step+1) = m_2(step)$
      $m_1(step+1) = l_2(step)$
      $m_2(step+1) = m_1(step)$
    ELSE
    % communication with $K - \frac{s_1}{2}$
      send $\begin{pmatrix} S_{l_1(step)}^{(k_0+step)} & R_{l_2(step)}^{(k_0+step)} \end{pmatrix}$ to proc $K - \frac{s_1}{2}$
      $m_1(step) = (K - \frac{s_1}{2}, 0)$

$m_2(step) = (K - \frac{s_1}{2}, N_0)$
receive $\begin{pmatrix} S^{(k_0+step)}_{m_1(step)} & R^{(k_0+step)}_{m_2(step)} \end{pmatrix}$ from proc $K - \frac{s_1}{2}$
$l_1(step+1) = m_1(step)$
$l_2(step+1) = l_2(step)$
$m_1(step+1) = m_2(step)$
$m_2(step+1) = l_1(step)$
ENDIF
$l = l + 1$

$$P_l \begin{pmatrix} R^{(k_0+step)}_{m_1(step+1)} \\ S^{(k_0+step)}_{m_2(step+1)} \end{pmatrix} = \begin{pmatrix} L_l \\ T_l \end{pmatrix} U_l$$

$$G_l^{(k_0+step)} = T_l L_l^{-1}$$

$$\begin{pmatrix} \widehat{S}^{(k_0+step)}_{l_1(step+1)} \\ S^{(k_0+step+1)}_{l_1(step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(k_0+step)} & I \end{pmatrix} P_l \begin{pmatrix} S^{(k_0+step)}_{l_1(step+1)} \\ O \end{pmatrix}$$

$$\begin{pmatrix} \widehat{R}^{(k_0+step)}_{l_2(step+1)} \\ R^{(k_0+step+1)}_{l_2(step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(k_0+step)} & I \end{pmatrix} P_l \begin{pmatrix} O \\ R^{(k_0+step)}_{l_2(step+1)} \end{pmatrix}$$

% update s_1 and $prod$
$s_1 = s_1 \cdot 2$
$prod = 1$
for $I = 0, \ldots, 2^{(step+1)} - 1$
$prod = prod \cdot mod(K - I, s_1)$
end
end
% Factorization of the final 2×2 system

$$\widehat{P} \begin{pmatrix} D_a & D_b \\ S_0^{(lnp+k_0)} & R_N^{(lnp+k_0)} \end{pmatrix} = \widehat{L}\widehat{U}$$

SOLUTION phase in proc K

% k_0 rhs reduction steps
for $step = 0, \ldots, k_0 - 1$
$s = 2^{step}$
for $j = 1, 2, \ldots, \lfloor \lceil N_0/s \rceil / 2 \rfloor$
% reduction of pairs of block rows
$l = (K, (2j-1)s)$

$$\begin{pmatrix} \widehat{\mathbf{f}}_l^{(step)} \\ \mathbf{f}_{l+s}^{(step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(step)} & I \end{pmatrix} P_i \begin{pmatrix} \mathbf{f}_l^{(step)} \\ \mathbf{f}_{l+s}^{(step)} \end{pmatrix}$$

end
end

$s_1 = 2$
$prod = mod(K, s_1)$
% lnp rhs communication steps
  for $step = 0, \ldots, lnp - 1$
    IF $(prod == 0)$ THEN
    % communication with $K + \frac{s_1}{2}$
      send $\mathbf{f}_{l_2(step)}^{(k_0+step)}$ to proc $K + \frac{s_1}{2}$
      receive $\mathbf{f}_{m_2(step)}^{(k_0+step)}$ from proc $K + \frac{s_1}{2}$
    ELSE
    % communication with $K - \frac{s_1}{2}$
      send $\mathbf{f}_{l_2(step)}^{(k_0+step)}$ to proc $K - \frac{s_1}{2}$
      receive $\mathbf{f}_{m_2(step)}^{(k_0+step)}$ from proc $K - \frac{s_1}{2}$
    ENDIF
    $l = l + 1$

$$\begin{pmatrix} \widehat{\mathbf{f}}_{m_1(step+1)}^{(k_0+step)} \\ \mathbf{f}_{l_2(step+1)}^{(k_0+step+1)} \end{pmatrix} = \begin{pmatrix} I & \\ -G_l^{(k_0+step)} & I \end{pmatrix} P_l \begin{pmatrix} \mathbf{f}_{m_1(step+1)}^{(k_0+step)} \\ \mathbf{f}_{l_2(step+1)}^{k_0+step} \end{pmatrix}$$

    % update s_1 and $prod$
    $s_1 = s_1 \cdot 2$
    $prod = 1$
    for $I = 0, \ldots, 2^{(step+1)} - 1$
      $prod = prod \cdot mod(K - I, s_1)$
    end
  end
% Solution of the final 2×2 system

$$\begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_N \end{pmatrix} = (\widehat{L}\widehat{U})^{-1}\widehat{P} \begin{pmatrix} \mathbf{d} \\ \mathbf{f}_N^{(lnp+k_0)} \end{pmatrix}$$

% Back substitution
  for $step = lnp - 1, \ldots, 0$
    $\mathbf{y}_{m_1(step)} = (L_l U_l)^{-1} \cdot$
      $\left(\widehat{\mathbf{f}}_{m_1(step)}^{(k_0+step)} - \widehat{S}_{l_1(step)}^{(k_0+step)} \mathbf{y}_{l_1(step)} - \widehat{R}_{l_2(step)}^{(k_0+step)} \mathbf{y}_{l_2(step)}\right)$
    $l = l - 1$
  end
  for $step = k_0 - 1, \ldots, 0$
    $s = 2^{step}$
    for $j = \lfloor \lceil N_0/s \rceil / 2 \rfloor, \ldots, 2, 1$
      $l = (K, (2j-1)s)$
      $\mathbf{y}_l = (L_l U_l)^{-1} \left(\widehat{\mathbf{f}}_l^{(step)} - \widehat{S}_{l-s}^{(step)} \mathbf{y}_{l-s} - \widehat{R}_{l+s}^{(step)} \mathbf{y}_{l+s}\right)$
    end
  end

5.3 Speedup

In Tables 5.1-5.2, we represent the speedup of the algorithm *PBABDCR*. That is, the ratio between the execution time (time of factorization plus time of solution) on one processor and the execution time on many processors. This implementation is applied to random BABD linear systems. We use a cluster of 16 (2.4 GHz Intel Xeon) processors with the Intel Fortran 90/95 compiler. Tables 5.1-5.2 describe the speedup of the algorithm, for fixed N and varying the dimension m of blocks and the number P of processors. These data are also represented in Figure 5.2 for $N = 512$ and in Figure 5.3 for $N = 1024$.

In Table 5.2 and in Figure 5.3, where $N = 1024$, we obtain higher speedup with increasing the dimension m for fixed P. In fact, since the factorization of an $m \times m$ matrix is the principal cost in each reduction, this cost (a timing) prevails over the time spent in the communication for increasing m. This results in a benefit of the multiprocessing procedure with respect the one processor version.

This happens only if the number of block rows, N, is sufficiently large with respect to m, especially when m is small (that yields low computational cost). In fact, for $m = 4$ and $N = 512$, see Table 5.1 and Figure 5.2, the speedup decreases if the number of processors increases. In this case the cost of communication prevails over the computational cost.

Wright, in [76], suggests a value of m that is at least 50 or 100 for obtaining better result of multiprocessing with respect to the one processor version.

Finally, for the results obtained in Section 3.8 in the serial case, we remark that *BABDCR* could result in a faster *MIRKDC* package on distributed parallel architectures than *PMIRKDC* [68]. We have not provided a direct comparison with *PMIRKDC* because it has been written for shared memory architectures and our linear solver *PBABDCR* can be implemented only on distributed memory architecture using MPI.

Table 5.1: Speedup of the code *PBABDCR* with $N = 512$.

Problem	$P = 2$	$P = 4$	$P = 8$	$P = 16$
$m = 4$	1.864	2.592	3.684	2.661
$m = 16$	1.800	3.715	6.590	10.412
$m = 64$	1.839	3.649	6.944	11.752

Table 5.2: Speedup of the code *PBABDCR* with $N = 1024$.

Problem	$P = 2$	$P = 4$	$P = 8$	$P = 16$
$m = 4$	1.917	2.951	4.508	6.306
$m = 16$	1.950	3.615	7.578	12.605
$m = 64$	2.030	3.912	7.768	14.153

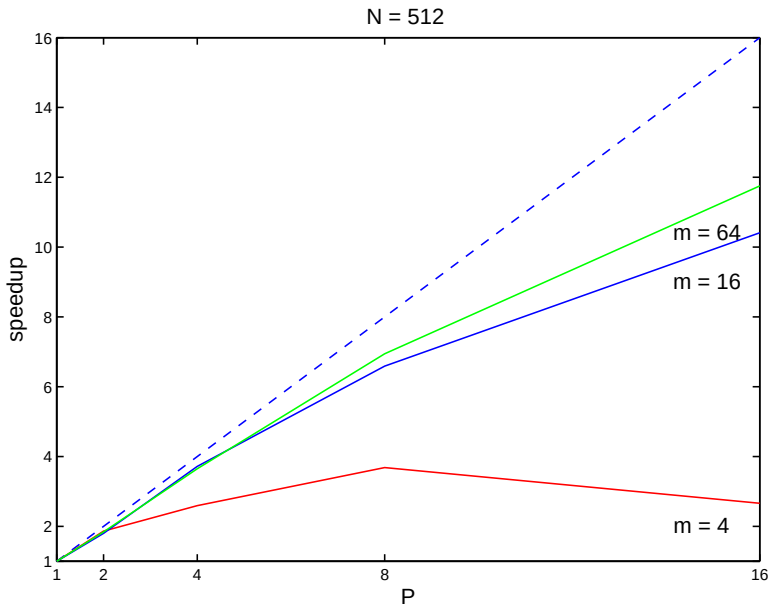

Figure 5.2: Speedup of *PBABDCR* with $N = 512$ and $P = 2, 4, 8, 16$ processors. The dashed line is the ideal speedup (speedup for P processors = P) that could be obtained, if the communications were instantaneous.

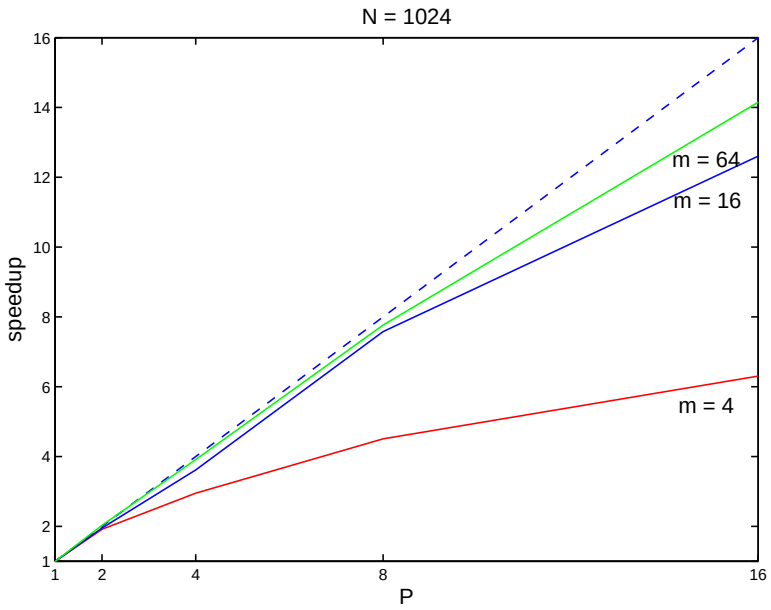

Figure 5.3: Speedup of *PBABDCR* with $N = 1024$ and $P = 2, 4, 8, 16$ processors. The dashed line is the ideal speedup (speedup for P processors = P) that could be obtained, if the communications were instantaneous.

Chapter 6

Appendix of Numerical Analysis

In this appendix, we state those basic facts about polynomials and Runge Kutta formulas needed in the previous chapters.

6.1 Polynomials

A *polynomial* of order $n \in \mathbb{N}$ or of degree at most $n-1$ is a function of the form

$$p(x) = a_1 + a_2 x + \ldots + a_n x^{n-1} = \sum_{j=1}^{n} a_j x^{j-1}$$

where a_j are real numbers, $a_j \in \mathbb{R}$. We use the letter $\mathcal{P}_n$ to denote the linear space of all polynomials of order n. We call *monomial* each polynomial $p(x) \in \mathcal{P}_n$ of type $p(x) = a_n x^{n-1}$. A sequence $\{p_i(x)\}_{i\geq 1}$ of *orthogonal polynomials* in $[a,b]$, with $p_i(x) \in \mathcal{P}_i$ has the property

$$\int_a^b p_i(x) p_j(x) dx = 0, \qquad i \neq j.$$

An example are the Legendre polynomials [17] with the kth *Legendre polynomial* in $[a,b]$

$$p_k(x) = \frac{(-1)^k}{(a+b)^k k!} \frac{d^k}{dx^k} \left(1 - \left(\frac{2}{a+b} x - 1 + \frac{2a}{a+b} \right)^2 \right)^k .$$

Given $k \in \mathbb{N}$, the *kth divided difference* of a function f at the points sequence $\{\tau_l\}_{l=1}^{k+1} = \{\tau_1, \ldots, \tau_{k+1}\}$ is the value

$$[\tau_1, \ldots, \tau_{k+1}]f := \begin{cases} \dfrac{f^{(k)}(\tau)}{k!} & \text{if } \tau = \tau_1 = \cdots = \tau_{k+1} \\ \dfrac{[\tau_1, \ldots, \tau_{i-1}, \tau_{i+1}, \ldots, \tau_{k+1}]f - [\tau_1, \ldots, \tau_{j-1}, \tau_{j+1}, \ldots, \tau_{k+1}]f}{\tau_j - \tau_i} & \\ \qquad \text{if } \tau_i, \tau_j \in \{\tau_l\}_{l=1}^{k+1} \text{ are any two distinct points } \tau_i \neq \tau_j \end{cases}$$

In the previous chapters, we use the property of the *kth divided difference*

$$[\tau_1, \ldots, \tau_{k+1}]g = 0$$

for any g polynomial of order k in $[\tau_1, \tau_{k+1}]$, see [18]. The *support* of a function $f(x)$, denoted as $supp(f)$ is the set

$$supp(f) = \{x \in \mathbb{R} | f(x) \neq 0\}.$$

We denote with $(x)_+$ the *"truncation" function*

$$(x)_+ := \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Other truncated functions are $(x)_+^j := x \cdot (x)_+^{j-1},\ j \in \mathbb{N}, j > 0$ and

$$(x)_+^0 := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Let $\Delta = \{x_i\}_{i=1}^{N+1}$ a *partition* or *subdivision* of the interval $[a, b]$ with strictly increasing sequence of points

$$\Delta : a = x_1 < x_2 < \cdots < x_N < x_{N+1} = b,$$

we call *piecewise polynomial* of order n defined on Δ, any function $f(x)$ such that $p_i(x) = f(x)|_{[x_i, x_{i+1}]}$ is a polynomial of order n, for $i = 1, \ldots, N$. We denote the collection of all such piecewise polynomial defined on Δ with $\mathcal{P}_{n,\Delta}$. An important class of piecewise polynomial is formed by the splines. We will use $\mathbb{S}_\Delta^{n,m}$ to denote the set of $\mathcal{C}^m$ *splines* of order n, $n \geq m$, defined on the partition Δ

$$\mathbb{S}_\Delta^{n,m} := \left\{ s(x) \in \mathcal{C}^m[a, b] : s(x)|_{[x_i, x_{i+1}]} \in \mathcal{P}_n, i = 1, \ldots, N \right\}.$$

$\lfloor x \rfloor$ denotes the greater integer less than or equal to x and $\lceil x \rceil$ denotes the smallest integer greater than or equal to x

6.2 Runge-Kutta formulae

The Runge-Kutta formulae [24] determine approximations of the solution of an IVODE problem in [a,b]

$$\begin{aligned} &\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)), \qquad \mathbf{y}(t) \in \mathbb{R}^m \qquad t \in [a, b] \\ &\mathbf{y}(t_0) = \mathbf{y}_0 \end{aligned}$$

$(t_0 \equiv a)$, using the scheme

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \sum_{r=1}^{s} b_r \mathbf{K}_{i,r}$$

with the stages

$$\mathbf{K}_{i,r} = \mathbf{f}(t_i + c_r h_i, \mathbf{y}_i + h_i \sum_{j=1}^{r} x_{rj}\mathbf{K}_{i,j}),\ r = 1, \ldots, s.$$

where s is the number of stages, $h_i = t_{i+1} - t_i$ is the amplitude of the ith subinterval of the given mesh $\{t_i\}_{i=0}^{N}$ composed by N subintervals. The given coefficients $\mathbf{b} = \{b_r\}_{r=1}^{s}$, called weight coefficients, the $s \times s$ lower triangular matrix $X := \{x_{r,j}\}_{r,j=1}^{s}$ and the ascissa vector $\mathbf{c} := \{c_r\}_{r=1}^{s}$, $0 \leq c_r \leq 1$, determines a unique Runge-Kutta formula which can be represented with the Butcher array

$$\begin{array}{c|c} \mathbf{c} & \mathbf{X} \\ \hline & \mathbf{b} \end{array}.$$

If for some r, $x_{rr} \neq 0$, then the RK formula is called *implicit*.

6.3 MIRK/CMIRK schemes in the code *MIRKDC*

Here we summarize the approach used in MIRKDC for employing a MIRK/CMIRK scheme applied to the BVP system (1.18)-(1.19) of dimension m. The approach consists of a two-level iteration scheme.

At beginning, the user determines an initial mesh (or partition) $\{t_i\}_{i=0}^{N}$ of N subintervals of $[a, b]$, an associated discrete solution approximation $\mathbf{Y}^{(0)}$ and a MIRK scheme, choosing a suitable Butcher array (1.17). The user defines also a defect tolerance TOL. The first level-step is the setup and the solution of the discrete system

$$\mathbf{\Phi}(\mathbf{Y}) = 0$$

where

$$\begin{array}{rl} \mathbf{Y} = (\mathbf{y}_0^T, \ldots, \mathbf{y}_N^T)^T,\ \mathbf{y}_j \in \mathbb{R}^m, & j = 0, \ldots, N, \\ \mathbf{\Phi}(\mathbf{Y}) = (\mathbf{\Phi}_0(\mathbf{Y})^T, \ldots, \mathbf{\Phi}_N(\mathbf{Y})^T)^T, & \mathbf{\Phi}_i : \mathbb{R}^{m(N+1)} \longrightarrow \mathbb{R}^m, \\ \mathbf{\Phi}_i(\mathbf{Y}) = \mathbf{y}_{i+1} - \mathbf{y}_i - h_i \sum_{r=1}^{s} b_r \mathbf{K}_{i,r}, & i = 0, \ldots, N-1, \\ \mathbf{\Phi}_N = \mathbf{g}(\mathbf{y}_0, \mathbf{y}_N). & \end{array}$$

with stages $\mathbf{K}_{i,r}$ defined in (1.16). The system is solved using the Newton iteration $\mathbf{Y}^{(q+1)} = \mathbf{Y}^{(q)} + \Delta\mathbf{Y}^{(q)}$, for $q = 0, 1, \ldots$, where

$$\left[\frac{\partial \mathbf{\Phi}(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}}\right] \Delta\mathbf{Y}^{(q)} = -\mathbf{\Phi}(\mathbf{Y}^{(q)}) \tag{6.1}$$

given $\mathbf{Y}^{(0)}$. On convergence the Newton iteration gives a discrete solution approximation $\widehat{\mathbf{Y}} = \left(\widehat{\mathbf{y}}_0^T, \ldots, \widehat{\mathbf{y}}_N^T\right)$.

Then, in the second step-level, we use an associated CMIRK scheme to costruct a C^1 continuous approximation solution over the entire interval $[a, b]$. The CMIRK scheme determine for each subinterval $[t_i, t_{i+1}]$ a polynomial in $[0, 1]$

$$\mathbf{u}(t_i + \theta h_i) = \widehat{\mathbf{y}}_i + h_i \sum_{r=1}^{s^*} b_r(\theta)\mathbf{K}_{i,r}, 0 \leq \theta \leq 1,\ t_i \leq t \leq t_{i+1}$$

with $s^* > s$ and $\mathbf{K}_{i,r}$, for $r = 1, \ldots, s$, are those defined in (1.16).

The extra stages

$$\mathbf{K}_{i,r} = \mathbf{f}(t_i + c_r h_i, (1 - v_r)\widehat{\mathbf{y}}_i + v_r \widehat{\mathbf{y}}_{i+1} + h_i \sum_{j=1}^{r-1} x_{rj}\mathbf{K}_{i,j}),\ r = s + 1, \ldots, s^*.$$

(that is, the remaining coefficients c_r,v_r and x_{rj}, $r = s+1, \ldots, s^*$, $j = 1, \ldots, r-1$) and the s^* polynomials $b_i(\theta)$ are chosen imposing on $\mathbf{u}(t)$, the continuous conditions on the current mesh and, for a given order p,

$$\max_{0 \leq \theta \leq 1} \|\mathbf{y}(t_i + \theta h_i) - \mathbf{u}(t_i + \theta h_i)\| = O(h_i^{p+1}), \quad i = 0, \ldots, N - 1,$$

where $\mathbf{y}(t)$ is the exact solution of the initial ODE $\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t))$ with $\mathbf{y}(t_i) = \mathbf{u}(t_i) = \widehat{\mathbf{y}}_i$. p is chosen as the order of accuracy of the discrete MIRK scheme chosen. Then, the approach proceeds with the estimate of the defect

$$\delta(t) := \mathbf{u}'(t) - \mathbf{f}(t, \mathbf{u}(t)), t \in [a, b]$$

on each subinterval δ_i. If the termination condition

$$\|\delta_i\| \leq TOL, \tag{6.2}$$

for $i = 0, \ldots, N - 1$ is not satisfied, then it uses the current mesh, defect estimates and TOL to construct new mesh which equidistributes defect estimates with sufficient refinement to approximately satisfy (6.2).

Finally $\mathbf{u}(t)$ is evaluated on the new mesh to set up the new discrete initial solution approximation, at the new mesh points. More details are described in [33] and [23].

Acknowledgements

I thank the staff of three Departments: the Department of Mathematics in Bari, the Department of Mathematics of the Southern Methodist University in Dallas, and the School of Computing in Leeds. I acknowledge, particularly, all my international colleagues for the technical and moral support during the three years spent in my PhD programme.

Giuseppe Romanazzi

Bari, February 2006

Bibliography

[1] P. Amodio, J. R. Cash, G. Roussos, R. W. Wright, G. Fairweather, I. Gladwell, G. L. Kraut, and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer. Linear Algebra Appl.*, 7:275–317, 2000.

[2] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput.*, 18(1):56–68, 1997.

[3] P. Amodio and G. Romanazzi. BABDCR: a Fortran 90 package for the solution of Bordered ABD systems. Rapporto 40/04, Dipartimento di Matematica, Università di Bari, Italy, 2004.

[4] P. Amodio and G. Romanazzi. Algorithm 8xx: BABDCR: a Fortran 90 package for the solution of bordered ABD linear systems. *ACM Transactions on Mathematical Software*, 2006.

[5] U.M. Ascher, J. Christiansen, and R.D. Russell. A collocation solver for mixed order systems of boundary value problems. *Math. Comp.*, 33:659–679, 1979.

[6] U.M. Ascher, J. Christiansen, and R.D. Russell. *COLSYS* — a collocation code for boundary value problems. In *Codes for Boundary Value Problems in Ordinary Differential Equations*, Springer-Verlag Lecture Notes in Computer Science 76, Childs B., Scott M., Daniel JW, Denman E, Nelson P. (eds.), pages 164–185. Springer-Verlag, New York, 1979. Code *colsys* is available from library *ode* on *netlib*.

[7] U.M. Ascher, J. Christiansen, and R.D. Russell. Algorithm 569: Colsys: Collocation software for boundary-value odes. *ACM Trans. Math. Softw.*, 7(2):223–229, 1981.

[8] U.M. Ascher, J. Christiansen, and R.D. Russell. Collocation software for boundary value ode's. *ACM Trans. Math. Softw.*, 7(2):209–222, 1981.

[9] U.M. Ascher, R.M.M. Mattheij, and R.D. Russell. *Numerical solution of boundary value problems for ordinary differential equations*. Classics in Applied Mathematics, 13. SIAM, Philadelphia, 2nd edition, 1995.

[10] U.M. Ascher, S. Pruess, and R.D. Russell. On spline basis selection for solving differential equations. *SIAM J. Numer. Anal.*, 20(1):121–142, 1983.

[11] U.M. Ascher and R.D. Russell. Reformulation of boundary value problems into 'standard form'. *SIAM J. Numer. Anal.*, 23(2):238–254, 1981.

[12] U.M. Ascher and R. Weiss. Collocation for singular perturbation problems I: First order systems with constant coefficients. *SIAM J. Numer. Anal.*, 20(3):537–557, 1983.

[13] G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value ode solver. *SIAM J. Sci. Statist. Comput.*, 8:483–500, 1987.

[14] F.W. Beaufait and G.W. Reddien. Midpoint difference method for analyzing beam structures. *Computers and Structures*, 8:745–751, 1978.

[15] R. Bellman and R. Kalaba. *Quasilinearization and Nonlinear Boundary Value Problems*. American Elsevier, New York, 1965.

[16] K.R. Bennett and G. Fairweather. A parallel boundary value ode code for shared memory machines. *International Journal of High Speed Computing*, 4:71–86, 1992. Code *pcolnew* is available by anonymous ftp from directory /pub/karin/pcolnew at math.nist.gov. (This version was prepared for specific shared memory parallel computers which are no longer widely available).

[17] R. Bevilacqua, D. Bini, M. Capovani, and O. Menchi. *Metodi numerici*. Zanichelli, Bologna, 1992.

[18] C. De Boor. *A practical guide to splines*. Springer-Verlag, New York, 1978.

[19] C De Boor and B. Swartz. Collocation at gaussian points. *SIAM J. Numer. Anal.*, 10(4):582–606, 1973.

[20] C. De Boor and R. Weiss. Algorithm 546. solveblok. *ACM Trans. Math. Softw.*, 6(1):88–91, 1980.

[21] C. De Boor and R. Weiss. Solveblok: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Softw.*, 6(1):80–87, 1980.

[22] R.W. Brankin and I. Gladwell. Codes for almost block diagonal systems. *Comput. Math. Appl.*, 19(7):1–6, 1990.

[23] K. Burrage, F.H. Chipman, and P.H. Muir. Order results for mono-implicit runge-kutta methods. *SIAM Journal on Numerical Analysis*, 31:876–891, 1994.

[24] J.C. Butcher. *The Numerical Analysis of Ordinary Differential Equations*. Wiley, Chichester, 1987.

[25] J.R. Cash, G. Moore, and R.W. Wright. An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems. *ACM Trans. Math. Software*, 27:245–266, 2001.

[26] J.R. Cash and R.W. Wright. A deferred correction method for nonlinear two-point boundary value problems: Implementations and numerical evaluation. *SIAM Journal of Scientific and Statistical Computing*, 12:971–989, 1991.

[27] C. Cyphers and M. Paprzycki. A level 3 blas based solver for almost block diagonal systems. SMU Softreport 92-3, Department of Mathematics, Southern Methodist University. Code *l*3abdsol is available from library *l*inalg on netlib, 1992.

[28] J.C. Diaz, G. Fairweather, and P. Keast. Algorithm 603: Colrow and arceco: Fortran packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Transactions on Mathematical Software*, 9(3):376–380, 1983.

[29] J.C. Diaz, G. Fairweather, and P. Keast. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Transactions on Mathematical Software*, 9:358–375, 1983.

[30] J. Dongarra, J. Du Croz, and S. Hammarling. A set of level 3 basic linear algebra subprograms. Technical report, ANL-MCS-TM57, Argonne National Laboratory, 1988. The level 3 BLAS are available on *n*etlib, in the NAG library, and, in optimized form, from many computer manufacturers.

[31] J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14:1–17, 1988. The level 2 BLAS are available on *n*etlib, in the NAG library, and, in optimized form, from many computer manufacturers.

[32] J. Dongarra, C.B. Moler, J.R. Bunch, and G.W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

[33] W.H. Enright and P.H. Muir. Runge-kutta software with defect control for boundary value odes. Technical Report 93-267, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1993.

[34] W.H. Enright and P.H. Muir. Runge-kutta software with defect control for boundary value odes. *SIAM J. Sci. Comput.*, 17:479–497, 1996.

[35] G. Fairweather and I. Gladwell. Algorithms for almost block diagonal linear systems. *SIAM Review*, 46(1):49–58, 2004.

[36] G. Fairweather and P. Keast. Rowcol– a package for solving almost block diagonal linear systems arising in h^{-1}-galerkin and collocation-h^{-1}-galerkin

methods. Technical Report 158/82, Department of Computer Science, University of Toronto, 1982.

[37] G. Fairweather, P. Keast, and J.C. Diaz. On the h^{-1}-galerkin method for second-order linear two-point boundary value problems. *SIAM J. Numer. Anal.*, 21:314–326, 1984.

[38] B. Garrett and I. Gladwell. Solving bordered almost block diagonal systems stably and efficiently. *J. Comput. Methods Sci. Engrg.*, 1:75–98, 2001.

[39] W. Gautschi. *Numerical analysis: an introduction*. Birkhäuser, Boston, 1997.

[40] I. Gladwell and R.I. Hay. Vector- and parallelization of ode bvp codes. *Parallel Comput.*, 12:343–350, 1989.

[41] I. Gladwell and M. Paprzycki. Solving almost block diagonal systems using level 3 blas. In *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, pages 52–62, Philadelphia, PA, 1992. SIAM.

[42] I. Gladwell and M. Paprzycki. Parallel solution of almost block diagonal systems on the cray y-mp using level 3 blas. *Journal of Computational and Applied Mathematics*, 45:181–189, 1993.

[43] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1983.

[44] W. Govaerts and J.D. Price. Mixed block elimination for linear system with wider borders. *IMA Journal of Numerical Analysis*, 13:161–180, 1993. Code *b*emw is available from library *l*inalg on *n*etlib.

[45] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer, Berlin, 2002.

[46] N. J. Higham. Fortran codes for estimating the one norm of a real or complex matrix with applications to condition number. Technical Report 135, University of Manchester, Manchester, 1987.

[47] N. J. Higham. Algorithm 674: FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Softw.*, 15(2):168, 1989.

[48] I.Gladwell and D.K. Sayers, editors. *Computational Techniques for Ordinary Differential Equations*. The Institute of Mathematics and its Applications, Academic Press, 1980.

[49] K.R. Jackson and R.N. Pancer. The parallel solution of abd systems arising in numerical methods for bvps for odes. Technical Report 255/91, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1992.

[50] P. Keast. Code *colrow.f*. It is available at http://www.mscs.dal.ca/∼keast/leq/colrow.f.

[51] P. Keast. Code *condcolrow.f*. It is available at http://www.mscs.dal.ca/∼keast/leq/condcolrow.f.

[52] P. Keast. Code *abdcnd.f*. It is available at http://www.mscs.dal.ca/∼keast/leq/abdcnd.f.

[53] P. Keast. Code *complexcolrow.f*. It is available at http://www.mscs.dal.ca/∼keast/leq/complexcolrow.f.

[54] P. Keast and R. Affleck. Codes for estimating condition numbers of certain almost block diagonal matrices. Technical report, Dalhousie University, Halifax, Nova Scotia B3H 3J5, Canada, 2000.

[55] P. Keast and G. Fairweather. Code *lampak.f*. It is available at http://www.mscs.dal.ca/∼keast/leq/[1997].

[56] P. Keast, G. Fairweather, and J.C. Diaz. A computational study of finite element methods for second order linear two-point boundary value problems. *Math. Comp.*, 40:499–518, 1983.

[57] P. Keast and P.H. Muir. Algorithm 688: Epdcol: a more efficient pdecol. *ACM Transactions on Mathematical Software*, 17:153–166, 1991.

[58] H.B. Keller. *Numerical Methods for Two Point Boundary Value Problems*. Dover, New York, 1992.

[59] D.C. Lam. *Implemantation of the box scheme and model analysis of diffusion–convenction equations*. PhD thesis, University of Waterloo, Waterloo, Canada, 1974.

[60] M. Lentini and V. Pereyra. An adaptive finite difference solver for nonlinear two-point boundary problems with mild boundary layers. *SIAM J. Numer. Anal.*, 14:91–111, 1977.

[61] M. Lentini and V. Pereyra. Pasva4: an ode boundary solver for problems with discontinous interfaces and algebraic parameters. *Math. Appl. Comput.*, 2:103–118, 1983.

[62] N.K. Madsen and R.F. Sincovec. Algorithm 540: Pdecol: general collocation software for partial differential equations. *ACM Transactions on Mathematical Software*, 5:326–351, 1979.

[63] F. Majaess, P. Keast, G. Fairweather, and K. R. Bennett. Algorithm 704: ABDPACK and ABBPACK-FORTRAN programs for the solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. *ACM Trans. Math. Softw.*, 18(2):205–210, 1992.

[64] F. Majaess, P. Keast, G. Fairweather, and K. R. Bennett. The solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. *ACM Trans. Math. Softw.*, 18(2):193–204, 1992.

[65] G. Micula and S. Micula. *Handbook of splines*. Kluwer, Dordrecht, 1999.

[66] P.H. Muir. Optimal discrete and continuous mono-implicit rungekutta schemes for bvodes. *Advances in Computational Mathematics*, 10:135167, 1999.

[67] P.H. Muir and B.Owren. Order barriers and characterizations for continuous mono-implicit runge-kutta schemes. *Mathematics of Computation*, 61:675–699, 1993.

[68] P.H. Muir, R.N. Pancer, and K.R. Jackson. Pmirkdc: a parallel mono-implicit rungekutta code with defect control for boundary value odes. *Parallel Computing*, 29:711741, 2003.

[69] NAG Ltd, Oxford, UK. *The NAG Fortran 77 Library*, 2000.

[70] T.B. Nokonechny, P. Keast, and P.H. Muir. A method of lines package, based on monomial spline collocation, for systems of one dimensional parabolic differential equations. In Griffiths D.F. and Watson G.A., editors, *Numerical Analysis, A.R. Mitchell 75th Birthday Volume*, pages 207–223, Singapore, 1993. World Scientific.

[71] S. M. Roberts and J. S. Shipman. *Two-point boundary value problems: Shooting Methods*. Elsevier Publishing Company, New York, 1972.

[72] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations*. Dover Publications, New York, 1985.

[73] M. van Veldhuizen. A note on partial pivoting and gaussian elimination. *Numerische Mathematik*, 29:1–10, 1977.

[74] J.M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.*, 13:71–75, 1976.

[75] R.W. Wright, J. Cash, and G. Moore. Mesh selection for stiff two-point boundary value problems. *Numer. Algorithms*, 7:205–224, 1994.

[76] S.J. Wright. Stable parallel algorithms for two-point boundary value problems. *SIAM J. Sci. Stat. Comput.*, 13(3):742–764, 1992.

[77] S.J. Wright. A collection of problems for which gaussian elimination with row partial pivoting is unstable. *SIAM J. Sci. Stat. Comp.*, 14:231–238, 1993.

[78] S.J. Wright and V. Pereyra. Adaption of a two point boundary value problems solver to a vector-multiprocessor environment. *SIAM J. Sci. Stat. Comput.*, 11:425–449, 1990.

[79] P.Y. Yalamov and M. Paprzycki. Stability and performance analysis of a block elimination solver for bordered linear systems. *IMA Journal of Numerical Analysis*, 19:335–348, 1999.