Numerical Linear Algebra with Applications, Vol. 1(1), 1–7 (93)

Almost Block Diagonal Linear Systems: Sequential and Parallel Solution Techniques, and Applications

P. Amodio

Dipartimento di Matematica, Università di Bari, I-70125 Bari, Italy

and

J. R. Cash, G. Roussos, R. W. Wright

Department of Mathematics, Imperial College, London SW7 2BZ, UK

and

G. Fairweather[1]

Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, CO 80401, USA

and

I. Gladwell[2]

Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

and

G. L. Kraut[3]

Department of Mathematics, The University of Texas at Tyler, Tyler, TX 75799, USA

and

1070–5325/93/010001–07$8.50

Received 29 September 1993
Revised 7 December 1993

Numerical Linear Algebra with Applications, Vol. 1(1), 1–7 (93)

M. Paprzycki

Department of Computer Science and Statistics, University of Southern Mississippi, Hattiesburg, MS 39406, USA

Almost block diagonal (ABD) linear systems arise in a variety of contexts, specifically in numerical methods for two-point boundary value problems for ordinary differential equations and in related partial differential equation problems. The stable, efficient sequential solution of ABDs has received much attention over the last fifteen years and the parallel solution more recently. We survey the fields of application with emphasis on how ABDs and bordered ABDs (BABDs) arise. We outline most known direct solution techniques, both sequential and parallel, and discuss the comparative efficiency of the parallel methods. Finally, we examine parallel iterative methods for solving BABD systems.

KEY WORDS Almost block diagonal systems, direct algorithms, iterative algorithms, parallel computing, boundary value problems, collocation methods, finite difference methods, multiple shooting methods

1. Introduction

In 1984, Fourer [72] gave a comprehensive survey of the occurrence of, and solution techniques for, linear systems with staircase coefficient matrices. Here, we specialize to a subclass of staircase matrices, almost block diagonal (ABD) matrices. We outline the origin of ABD matrices in solving ordinary differential equations (ODEs) with separated boundary conditions (BCs) (that is, boundary value ordinary differential equations (BVODEs)) and provide a survey of old and new algorithms for solving ABD linear systems, including some for parallel implementation. Also, we discuss bordered ABD (BABD) systems which arise in solving BVODEs with nonseparated BCs, and describe a variety of solution techniques.

The most general ABD matrix [32], shown in Figure 1..1, has the following characteristics: the nonzero elements lie in blocks which may be of different sizes; each diagonal entry lies in a block; any column of the matrix intersects no more than two blocks (which are successive), and the overlap between successive blocks (that is, the number of columns of the matrix common to two successive blocks) need not be constant. In commonly used methods for solving BVODEs with separated BCs, the most frequently occurring ABD structure is shown in Figure 1..2, where the blocks $W^{(i)}$, $i = 1, 2, \ldots, N$, are all of equal size, and the overlap between successive blocks is constant and equal to the sum of the number of rows in *TOP* and *BOT*. Over the last two decades, this special structure has been exploited in a number of algorithms to minimize fill-in and computational cost without compromising stability. Naive approaches to solving ABD systems involve considering them as banded or block tridiagonal systems. These approaches are undesirable for several reasons not the least of which is that they introduce fill-in when the structure is imposed on the system as well as in the solution procedure, leading to significant ineffiencies.

[1] This work was supported in part by National Science Foundation grants CCR-9403461 and DMS-9805827
[2] This work was supported in part by NATO grant CRG 920037
[3] Current address: Department of Mathematics, University of New Hampshire, Durham, NH 03824

1070–5325/93/010001–07$8.50

Received 29 September 1993
Revised 7 December 1993

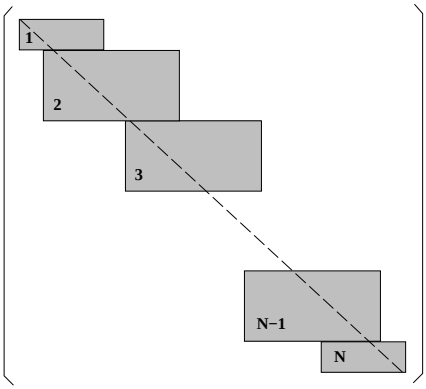

Figure 1..1. Structure of a general ABD matrix

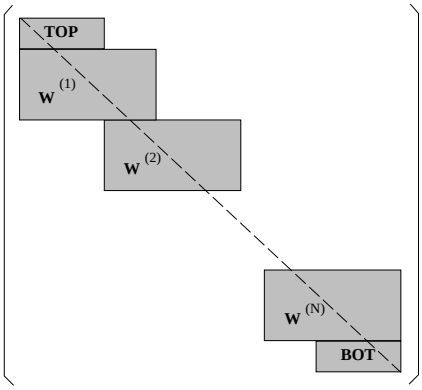

Figure 1..2. Special ABD structure arising in BVODE solvers

A BABD matrix differs from the ABD matrix of Figure 1..1 in its last block row (or column), where at least the first block entry is nonzero; for example, see the matrix in (2..15) following. In some applications, the whole of the last block row (or column or both) of the BABD is nonzero; for example, see the matrix in (2..20) following.

In Section 2., we outline how ABD and BABD systems arise when solving BVODEs using three important basic techniques: finite differences, multiple shooting and orthogonal spline collocation (OSC). This material is introductory and is included here for completeness. In Section 3., we provide a comprehensive survey of the origin of systems of similar structure in finite difference and OSC techniques for elliptic boundary value problems and initial-boundary value problems. Next, in Section 4., we describe efficient sequential direct solution techniques for ABD and BABD systems. Emphasis is placed on algorithms based on the alternate row and column elimination scheme of Varah [142]. As described in Subsection 4.1.6. below, these algorithms are employed in modern software packages for solving differential equations. In Section 5., we outline a variety of direct solution tech-

niques for parallel implementation. The emphasis is on BABD systems as the techniques derived are effective for both ABD and BABD systems. Here, we give some theoretical arithmetic and communication costs, and storage counts for sequential and parallel implementations. Finally, in Section 6., we describe iterative techniques including comments on their parallel.

2. BVODEs and ABD Systems

A general first order system of nonlinear BVODEs with separated BCs has the form

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)),\ x \in [a, b],\ \ \mathbf{g}_a(\mathbf{y}(a)) = \mathbf{0},\ \mathbf{g}_b(\mathbf{y}(b)) = \mathbf{0}, \tag{2..1}$$

where $\mathbf{y}, \mathbf{f} \in \mathcal{R}^n$, $\mathbf{g}_a \in \mathcal{R}^q$, $\mathbf{g}_b \in \mathcal{R}^{n-q}$. Among the most widely used methods for its numerical solution are finite differences [11,95,104,105], multiple shooting [11,95], and OSC [9,10,14]. When any of these methods are implemented using a quasilinearization or Newton method, each iteration of the method requires the solution of an ABD system of linear algebraic equations. This similarity is not surprising since the three approaches have a close relationship [103]. In particular, many finite difference schemes can be interpreted as multiple shooting algorithms where the shooting is over just one mesh subinterval using an appropriate one step initial value problem (IVP) method. Also, OSC is equivalent to using a (Gauss) Implicit Runge-Kutta (IRK) one step scheme [143,144] when applied to a BVODE in the form (2..1) (but not when applied to higher order systems of BVODEs).

Consider the linear BVODE

$$\mathbf{y}'(x) = A(x)\mathbf{y}(x) + \mathbf{r}(x),\ x \in [a, b],\ \ D_a\mathbf{y}(a) = \mathbf{c}_a,\ D_b\mathbf{y}(b) = \mathbf{c}_b, \tag{2..2}$$

where $\mathbf{y}(x), \mathbf{r}(x) \in \mathcal{R}^n$, $A(x) \in \mathcal{R}^{n\times n}$, $D_a \in \mathcal{R}^{q\times n}$, $D_b \in \mathcal{R}^{(n-q)\times n}$, $\mathbf{c}_a \in \mathcal{R}^q$, $\mathbf{c}_b \in \mathcal{R}^{(n-q)}$. In the solution of the nonlinear BVODE (2..1), problem (2..2) arises at each iteration of a quasilinearization method. We simply replace $A(x)$ by $\partial\mathbf{f}(x, \hat{\mathbf{y}}(x))/\partial\mathbf{y}$, the Jacobian of $\mathbf{f}(x, \mathbf{y}(x))$, $\mathbf{r}(x)$ by $\partial\mathbf{f}(x, \hat{\mathbf{y}}(x))/\partial x$, and D_a, D_b by the Jacobians of the BCs, $\partial\mathbf{g}_a(\hat{\mathbf{y}}(a))/\partial\mathbf{y}(a)$, $\partial\mathbf{g}_b(\hat{\mathbf{y}}(b))/\partial\mathbf{y}(b)$, respectively. (Here, $\hat{\mathbf{y}}(x)$ is the previous iterate for the solution in the quasilinearization process.)

To describe the basic methods in this and later sections, we choose a mesh

$$\pi : a = x_0 < x_1 < \ldots < x_N = b, \quad h_i = x_i - x_{i-1}. \tag{2..3}$$

2.1. *Finite Difference Methods*

Finite difference schemes can be illustrated using the one step trapezoidal method,

$$\mathbf{y}_i - \mathbf{y}_{i-1} = \frac{h_i}{2}[\mathbf{f}(x_{i-1}, \mathbf{y}_{i-1}) + \mathbf{f}(x_i, \mathbf{y}_i)], \tag{2..4}$$

where $\mathbf{y}_i \approx \mathbf{y}(x_i)$. Applying (2..4) to (2..2) for $i = 1, 2, \ldots, N$, gives

$$\begin{pmatrix} D_a & & & & \\ S_1 & T_1 & & & \\ & S_2 & T_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & T_N \\ & & & & D_b \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{c}_a \\ h_1\mathbf{r}_{\frac{1}{2}} \\ h_2\mathbf{r}_{\frac{3}{2}} \\ \vdots \\ h_N\mathbf{r}_{N-\frac{1}{2}} \\ \mathbf{c}_b \end{pmatrix}, \tag{2..5}$$

where $S_i, T_i \in \mathcal{R}^{n\times n}$, $\mathbf{r}_{i-\frac{1}{2}} \in \mathcal{R}^n$ are

$$S_i = -I - \frac{h_i}{2}A(x_{i-1}),\quad T_i = I - \frac{h_i}{2}A(x_i),\quad \mathbf{r}_{i-\frac{1}{2}} = \frac{1}{2}[\mathbf{r}(x_{i-1}) + \mathbf{r}(x_i)],\qquad i = 1, 2, \ldots, N.$$

In the following, we refer to the system (2..5) as being in standard ABD form.

2.2. Multiple Shooting

In multiple shooting for (2..2), we compute a fundamental solution matrix $Y_i(x) \in \mathcal{R}^{n\times n}$ and a particular solution $\mathbf{p}_i(x) \in \mathcal{R}^n$ on each subinterval $[x_{i-1}, x_i],\ i = 1, 2, \ldots, N$, of the mesh π:

$$Y_i'(x) = A(x)Y_i(x),\quad Y_i(x_{i-1}) = I,\qquad \mathbf{p}_i' = A(x)\mathbf{p}_i + \mathbf{r}(x),\quad \mathbf{p}_i(x_{i-1}) = \mathbf{0}. \quad (2..6)$$

Then the vectors $\mathbf{s}_i \in \mathcal{R}^n$ in

$$\mathbf{y}(x) = Y_i(x)\mathbf{s}_i + \mathbf{p}_i(x),\qquad x \in [x_{i-1}, x_i],\quad i = 1, 2, \ldots N,$$

are chosen to preserve continuity of the approximate solution at the mesh points. Thus,

$$Y_i(x_i)\mathbf{s}_i + \mathbf{p}_i(x_i) = Y_{i+1}(x_i)\mathbf{s}_{i+1} + \mathbf{p}_{i+1}(x_i),\quad i = 1, 2, \ldots N-1.$$

Because of the initial conditions, it follows that

$$-Y_i(x_i)\mathbf{s}_i + \mathbf{s}_{i+1} = \mathbf{p}_i(x_i),\qquad i = 1, 2, \ldots, N-1.$$

Combining these equations with the BCs in (2..2) gives

$$\begin{pmatrix} D_aY_1(x_0) & & & & \\ -Y_1(x_1) & I & & & \\ & -Y_2(x_2) & I & & \\ & \ddots & \ddots & & \\ & & -Y_{N-1}(x_{N-1}) & I \\ & & & D_bY_N(x_N) \end{pmatrix} \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \\ \vdots \\ \mathbf{s}_{N-1} \\ \mathbf{s}_N \end{pmatrix} = \begin{pmatrix} \mathbf{c}_a \\ \mathbf{p}_1(x_1) \\ \mathbf{p}_2(x_2) \\ \vdots \\ \mathbf{p}_{N-1}(x_{N-1}) \\ \mathbf{c}_b - D_b\mathbf{p}_N(x_N) \end{pmatrix} \quad (2..7)$$

which is an ABD system with a special structure that can be exploited.

2.3. Orthogonal Spline Collocation Methods

To simplify the exposition, we present OSC using Lagrangean basis functions for problem (2..2). Then we give more details in the context of second order scalar linear BVODEs with separated BCs which leads into the discussion in Section 3..

On each subinterval $[x_{i-1}, x_i],\ i = 1, 2, \ldots, N$, of the mesh π, the aim is to find a polynomial $\mathbf{P}_i(x)$ of degree no greater than $r-1$ ($r \geq 2$) of the form

$$\mathbf{P}_i(x) = \sum_{j=0}^{r-1} \mathbf{y}_{i-1+j/(r-1)} \prod_{\substack{l=0 \\ l\neq j}}^{r-1} \frac{x - x_{i-1+l/(r-1)}}{x_{i-1+j/(r-1)} - x_{i-1+l/(r-1)}},$$

where $x_{i-1+j/(r-1)} = x_{i-1} + jh_i/(r-1)$. This polynomial must satisfy

$$\mathbf{P}_i'(\xi_{(i-1)(r-1)+k}) = A(\xi_{(i-1)(r-1)+k})\mathbf{P}_i(\xi_{(i-1)(r-1)+k}) + \mathbf{r}(\xi_{(i-1)(r-1)+k}),$$

where $\xi_{(i-1)(r-1)+k} = x_{i-1} + h_i\sigma_k$, $k = 1, 2, \ldots, r-1$, are the Gauss points on $[x_{i-1}, x_i]$, $i = 1, 2, \ldots, N$, and $\{\sigma_k\}_{k=1}^{r-1}$ are the Gauss Legendre nodes on $[0, 1]$ corresponding to the zeros of the Legendre polynomial of degree $r-1$, and the BCs

$$D_a\mathbf{P}_1(a) = \mathbf{c}_a, \quad D_b\mathbf{P}_N(b) = \mathbf{c}_b.$$

Thus, there are $[(r-1)N + 1]n$ linear equations arising from the collocation conditions and the BCs, and there are $(N-1)n$ linear equations arising from requiring continuity of the polynomials $\mathbf{P}_i(x)$ at the interior mesh points x_i, $i = 1, 2, \ldots, N-1$, of π. Using "condensation", we eliminate the unknowns at the non-mesh points. The resulting linear system has $(N+1)n$ equations in standard ABD form (2..5).

Next, consider the linear second order scalar BVODE with separated BCs:

$$Lu \equiv -a(x)u'' + b(x)u' + c(x)u = f(x), \quad x \in [a, b], \tag{2..8}$$

$$\alpha_a u(a) + \beta_a u'(a) = g_a, \quad \alpha_b u(b) + \beta_b u'(b) = g_b. \tag{2..9}$$

For the mesh π of (2..3), let

$$\mathcal{M}_r(\pi) = \{v \in C^1[a, b] : v|_{[x_{i-1}, x_i]} \in P_r, i = 1, 2, \ldots, N\},$$

where P_r denotes the set of all polynomials of degree $\leq r$. Also, let

$$\mathcal{M}_r^0(\pi) = \mathcal{M}_r(\pi) \cap \{v | v(a) = v(b) = 0\}.$$

Note that

$$\dim(\mathcal{M}_r^0(\pi)) \equiv M = N(r-1), \quad \dim(\mathcal{M}_r(\pi)) = M + 2.$$

In the OSC method for (2..8)-(2..9), the approximate solution $U \in \mathcal{M}_r(\pi)$, $r \geq 3$. If $\{\phi_j\}_{j=1}^{M+2}$ is a basis for $\mathcal{M}_r(\pi)$, we may write

$$U(x) = \sum_{j=1}^{M+2} u_j\phi_j(x),$$

and $\{u_j\}_{j=1}^{M+2}$ is determined by requiring that U satisfy (2..8) at $\{\xi_j\}_{j=1}^{M}$, and the BCs (2..9):

$$\begin{aligned} \alpha_a U(a) + \beta_a U'(a) &= g_a, \\ LU(\xi_j) &= f(\xi_j), \quad j = 1, 2, \ldots, M, \\ \alpha_b U(b) + \beta_b U'b) &= g_b. \end{aligned}$$

First, consider the special case $r = 3$, for which the collocation points are

$$\xi_{2i-1} = \frac{1}{2}(x_{i-1} + x_i) - \frac{1}{2\sqrt{3}}h_i, \quad \xi_{2i} = \frac{1}{2}(x_{i-1} + x_i) + \frac{1}{2\sqrt{3}}h_i, \quad i = 1, 2, \ldots, N.$$

With the usual basis $\{v_j\}_{j=0}^N \cup \{s_j\}_{j=0}^N$ for $\mathcal{M}_3(\pi)$ [63], we set

$$U(x) = \sum_{l=0}^{N} \{u_l v_l(x) + u_l' s_l(x)\}.$$

Since only four basis functions, v_{i-1}, s_{i-1}, v_i, s_i, are nonzero on $[x_{i-1}, x_i]$, the coefficient matrix of the collocation equations is ABD of the form (2..5) with $D_a = [\alpha_a \quad \beta_a]$, $D_b = [\alpha_b \quad \beta_b]$, and

$$S_i = \begin{pmatrix} Lv_{i-1}(\xi_{2i-1}) & Ls_{i-1}(\xi_{2i-1}) \\ Lv_{i-1}(\xi_{2i}) & Ls_{i-1}(\xi_{2i}) \end{pmatrix}, \quad T_i = \begin{pmatrix} Lv_i(\xi_{2i-1}) & Ls_i(\xi_{2i-1}) \\ Lv_i(\xi_{2i}) & Ls_i(\xi_{2i}) \end{pmatrix}, \quad i = 1, 2, \ldots, N.$$

For $r > 3$, with a standard ordering of the B-spline basis as implemented in the code *colsys* [9,10], the coefficient matrix

$$\begin{pmatrix} D_a & & & & & & & \\ W_{11} & W_{12} & W_{13} & & & & & \\ & & W_{21} & W_{22} & W_{23} & & & \\ & & & & \ddots & & & \\ & & & & W_{N1} & W_{N2} & W_{N3} & \\ & & & & & & D_b \end{pmatrix} \tag{2..10}$$

is ABD, with $W_{i1} \in \mathcal{R}^{(r-1)\times 2}, W_{i2} \in \mathcal{R}^{(r-1)\times(r-3)}, W_{i3} \in \mathcal{R}^{(r-1)\times 2}$; see [33].

For monomial basis functions, as in *colnew* [12,14], the linear system, of order $N(r+1)+2$, is

$$\begin{pmatrix} D_a & & & & & & & & \\ V_1 & W_1 & & & & & & & \\ -C_1 & -D_1 & I & & & & & & \\ & & \ddots & & & & & & \\ & & V_i & W_i & & & & & \\ & & -C_i & -D_i & I & & & & \\ & & & & \ddots & & & & \\ & & & & V_N & W_N & & & \\ & & & & -C_N & -D_N & I & & \\ & & & & & & & D_b \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{z}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{i-1} \\ \mathbf{z}_{i-1} \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{z}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} g_a \\ \mathbf{q}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{q}_i \\ \mathbf{0} \\ \vdots \\ \mathbf{q}_N \\ \mathbf{0} \\ g_b \end{pmatrix}, \tag{2..11}$$

where $V_i \in \mathcal{R}^{(r-1)\times 2}, W_i \in \mathcal{R}^{(r-1)\times(r-1)}, C_i \in \mathcal{R}^{2\times 2}, D_i \in \mathcal{R}^{2\times(r-1)}$. Often, such systems are first condensed by eliminating the internal variables $\mathbf{z}_i$. For suitably small h_i, W_i is nonsingular [14] and we have

$$\mathbf{z}_{i-1} = W_i^{-1}\mathbf{q}_i - W_i^{-1}V_i\mathbf{y}_{i-1}, \quad i = 1, 2, \ldots, N,$$

which, when substituted in

$$\mathbf{y}_i = C_i\mathbf{y}_{i-1} + D_i\mathbf{z}_{i-1}, \quad i = 1, 2, \ldots, N,$$

gives the condensed equations

$$\Gamma_i\mathbf{y}_{i-1} - \mathbf{y}_i = -D_iW_i^{-1}\mathbf{q}_i, \quad \Gamma_i = C_i - D_iW_i^{-1}V_i, \quad \Gamma_i \in \mathcal{R}^{2\times 2}, \quad i = 1, 2, \ldots, N. \tag{2..12}$$

Thus, (2..11) is reduced to an ABD system of order $2(N+1)$, cf. (2..7),

$$\begin{pmatrix} D_a & & & & & & \\ -\Gamma_1 & I & & & & & \\ & \ddots & \ddots & & & & \\ & & -\Gamma_i & I & & & \\ & & & \ddots & \ddots & & \\ & & & & -\Gamma_N & I & \\ & & & & & D_b \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{i-1} \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} g_a \\ D_1 W_1^{-1} \mathbf{q}_1 \\ \vdots \\ D_i W_i^{-1} \mathbf{q}_i \\ \vdots \\ D_N W_N^{-1} \mathbf{q}_N \\ g_b \end{pmatrix}. \quad (2..13)$$

Condensation accounts for a significant proportion of the total execution time in *colnew* [75]. Computing each Γ_i in (2..13) requires factoring W_i and solving two linear systems (for an m^{th} order problem, m systems), and computing $DW_i^{-1}\mathbf{q}_i$ requires solving one linear system. Generating each block system (2..12) is completely independent. So, both the generation and the solution phases of condensation have a highly parallel structure, as exploited in the code *pcolnew* [15,16].

2.4. Special BVODEs

We consider generalizations of (2..1) which, when solved by standard methods, also give rise to ABD or BABD linear systems.

2.4.1. Nonseparated BCs Here, the BCs in (2..1) are replaced by

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}, \quad \mathbf{g} \in \mathcal{R}^n. \quad (2..14)$$

For any standard discretization, the BABD system associated with BCs (2..14) has coefficient matrix

$$\begin{pmatrix} S_1 & T_1 & & & \\ & S_2 & T_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & T_N \\ B_a & & & & B_b \end{pmatrix}, \quad (2..15)$$

where B_a, B_b are Jacobians of $\mathbf{g}$ with respect to $\mathbf{y}(a), \mathbf{y}(b)$, respectively. In the linear case,

$$B_a \mathbf{y}(a) + B_b \mathbf{y}(b) = \mathbf{c}, \quad B_a, B_b \in \mathcal{R}^{n \times n}, \ \mathbf{c} \in \mathcal{R}^n. \quad (2..16)$$

This structure and an associated special solution technique were derived for the first high-quality automatic BVODE solver *pasva3*, based on finite differences [104,105]; see [150] for details of vectorization of the linear algebra in this solver. This approach can be extended straightforwardly to one step IVP schemes such as IRK methods and to special subclasses of IRK methods such as the mono-implicit Runge-Kutta (MIRK) methods, first introduced in [40]; see [36,37,41,42,61,62,81,120,121] for more on the use of IRK methods for BVODEs.

For multiple shooting, for the BCs (2..16), the linear BABD system corresponding to the

ABD system (2..20) for non-separated BCs is

$$\begin{pmatrix} -Y_1(x_1) & I & & & \\ & -Y_2(x_2) & I & & \\ & \ddots & \ddots & & \\ & & -Y_{N-1}(x_{N-1}) & I & \\ B_a & & & B_bY_N(x_N) & \end{pmatrix} \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_{N-1} \\ \mathbf{s}_N \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1(x_1) \\ \mathbf{p}_2(x_2) \\ \vdots \\ \mathbf{p}_{N-1}(x_{N-1}) \\ \mathbf{c}_b - B_b\mathbf{p}_N(x_N) \end{pmatrix} \quad (2..17)$$

The BABD system (2..17) is discussed in detail in [11, page 150], where it is shown that, for N sufficiently large, it is well-conditioned when the underlying BVODE (2..14) is well-conditioned.

In a closely related case, the BCs are partially separated; that is, they have the form

$$\mathbf{g}_a(\mathbf{y}(a)) = \mathbf{0}, \mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}. \quad (2..18)$$

The associated system has coefficient matrix

$$\begin{pmatrix} D_a & & & & \\ S_1 & T_1 & & & \\ & S_2 & T_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & T_N \\ \tilde{B}_a & & & & \tilde{B}_b \end{pmatrix}, \quad (2..19)$$

where $\tilde{B}_a, \tilde{B}_b \in \mathcal{R}^{(n-q)\times n}$ are Jacobians of $\mathbf{g}$ with respect to $\mathbf{y}(a)$, $\mathbf{y}(b)$, respectively.

Using a technique described in [13], we can convert nonseparated, or partially separated, BCs into separated form. For example, in the nonseparated case, we add n trivial equations and associated initial conditions:

$$\mathbf{z}_i' = \mathbf{0}, \quad \mathbf{z}_i(a) = \mathbf{y}_i(a), \qquad i = 1, 2, \ldots, n,$$

giving a system of size $2n$ with separated BCs:

$$\begin{array}{ll} \mathbf{y}' = \mathbf{f}(x, \mathbf{y}), & \mathbf{g}(\mathbf{z}(b), \mathbf{y}(b)) = \mathbf{0}, \\ \mathbf{z}' = \mathbf{0}, & \mathbf{z}(a) = \mathbf{y}(a). \end{array}$$

Then, any standard numerical technique will lead to a system with an ABD coefficient matrix of type (2..5) but with submatrices of order $2n$. Partially separated BCs can be treated similarly.

2.4.2. Multipoint conditions Next, consider linear multipoint conditions

$$\sum_{j=0}^{N} B_j\mathbf{y}(x_j) = \mathbf{c}, \quad B_j \in \mathcal{R}^{n\times n}, \quad \mathbf{c} \in \mathcal{R}^n,$$

where the points x_j coincide with mesh points. Normally only a few B_j will be nonzero; that is, there will be mesh points x_i which are not multipoints. A first order BVODE system with these BCs when solved by a standard method gives rise to a BABD system with

coefficient matrix

$$\begin{pmatrix} S_1 & T_1 & & & & \\ & S_2 & T_2 & & & \\ & & \ddots & \ddots & & \\ & & & S_N & T_N & \\ B_0 & B_1 & \dots & B_{N-1} & B_N \end{pmatrix}, \tag{2..20}$$

where B_0, B_N and some of $B_1,\ B_2, \dots,\ B_{N-1}$ are nonzero. It is possible to convert this system into several systems of two-point BVODEs with separated BCs, as explained in [11, page 6].

2.4.3. Parameter Dependent Problems and Integral Constraints Parameters may appear in the BVODE. Consider the linear (in $\mathbf{y}$) case:

$$\mathbf{y}' = A(x, \boldsymbol{\lambda})\mathbf{y} + \mathbf{r}(x, \boldsymbol{\lambda}), \quad B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{c}(\boldsymbol{\lambda}), \tag{2..21}$$

where $\mathbf{y} \in \mathcal{R}^n,\ \boldsymbol{\lambda} \in \mathcal{R}^m,\ B_a, B_b \in \mathcal{R}^{(m+n)\times n}$. We need the $m + n$ BCs so that both $\mathbf{y}$ and $\boldsymbol{\lambda}$ may be determined. The coefficient matrix is

$$\begin{pmatrix} S_1 & T_1 & & & & Z_1 \\ & S_2 & T_2 & & & Z_2 \\ & & \ddots & \ddots & & \vdots \\ & & & S_N & T_N & Z_N \\ B_a & & & & B_b & Z_c \end{pmatrix}, \tag{2..22}$$

where Z_c depends on $\mathbf{c}(\boldsymbol{\lambda})$, and $Z_1, Z_2, \dots, Z_N$ on $A(x, \boldsymbol{\lambda})$ and $\mathbf{r}(x, \boldsymbol{\lambda})$. Instead, we could deal with this problem by adding $\boldsymbol{\lambda}' = \mathbf{0}$ to yield a standard BVODE in $m + n$ unknowns, essentially as in Section 2.4.1..

An important extension of the standard BVODE (2..1) involves integral constraints. Here, we consider the more general case with parameters. With $\lambda \in \mathcal{R}$ fixed and given, the ODEs, BCs and integral constraints are

$$\begin{aligned} \mathbf{y}' = \mathbf{f}(x, \mathbf{y}, \boldsymbol{\tau}, \lambda), \quad x \in [a, b], \qquad & \mathbf{y}, \mathbf{f} \in \mathcal{R}^n, \quad \boldsymbol{\tau} \in \mathbf{R}^{n_\tau}, \\ \mathbf{g}(\mathbf{y}(a), \mathbf{y}(b), \boldsymbol{\tau}, \lambda) = \mathbf{0}, \qquad & \mathbf{g} \in \mathcal{R}^{n_g}, \\ \int_a^b \mathbf{w}(t, \mathbf{y}(t), \boldsymbol{\tau}, \lambda)dt = \mathbf{0}, \qquad & \mathbf{w} \in \mathcal{R}^{n_w}, \end{aligned} \tag{2..23}$$

which we must solve for $\mathbf{y}$, $\boldsymbol{\tau}$. Clearly, we require $n_g + n_w = n + n_\tau$ for the problem to be well-posed.

Important practical applications related to (2..23) occur in bifurcation analysis, for example, in computing periodic orbits, Hopf bifurcations and heteroclinic orbits [96,118,119]; see also [18] for an engineering application. As an example, consider computing periodic orbits. The parameter dependent (autonomous) ODE is

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, \lambda), \quad \mathbf{y}, \mathbf{f} \in \mathbf{R}^n,\ \lambda \in \mathcal{R}, \tag{2..24}$$

where λ is given but the solution of (2..24) is of unknown period; a well-known example of (2..24) is the system of Lorenz equations [118]. The usual first step is to transform the independent variable to $[0, 1]$ so that the period X now appears explicitly as an unknown:

$$\mathbf{y}^{\{1\}} = X\mathbf{f}(\mathbf{y}, \lambda), \quad \mathbf{y}, \mathbf{f} \in \mathbf{R}^n, \quad X, \lambda \in \mathcal{R}, \quad \mathbf{y}(0) = \mathbf{y}(1), \tag{2..25}$$

where the independent variable is a scaled version of time and $\{1\}$ denotes differentiation with respect to the transformed variable. (An alternative formulation which also gives rise to BABD systems is described in [118].) Suppose we have computed a periodic solution with a given $\lambda = \lambda_{j-1}$, say $(\mathbf{y}_{j-1}, X_{j-1}, \lambda_{j-1})$, and we wish to compute another periodic solution $(\mathbf{y}_j, X_j, \lambda_j)$. Since $\mathbf{y}(x)$ is periodic, so is $\mathbf{y}(x+\delta)$ for any δ. Thus, we need a phase condition, for example

$$\int_0^1 \mathbf{y}(x)^T \mathbf{y}_{j-1}^{\{1\}}(x)dx = 0, \tag{2..26}$$

to specify the solution completely. We also add the pseudo-arclength equation

$$\int_0^1 (\mathbf{y}(s) - \mathbf{y}_{j-1}(s))^T \dot{\mathbf{y}}_{j-1}(s)ds + (X - X_{j-1})\dot{X}_{j-1} + (\lambda - \lambda_{j-1})\dot{\lambda}_{j-1} = \Delta s. \tag{2..27}$$

Here, λ is regarded as unknown and Δs as given, and the dot denotes differentiation with respect to arclength. Assuming Δs constant, we might use the approximations

$$\dot{X}_{j-1} = \frac{X_{j-1} - X_{j-2}}{\Delta s}, \quad \dot{\lambda}_{j-1} = \frac{\lambda_{j-1} - \lambda_{j-2}}{\Delta s}, \quad \dot{\mathbf{y}}_{j-1} = \frac{\mathbf{y}_{j-1} - \mathbf{y}_{j-2}}{\Delta s}. \tag{2..28}$$

When equations (2..25)-(2..28) are discretized using standard methods, for each Newton iterate, the linear system has a BABD coefficient matrix,

$$\mathcal{A} = \begin{pmatrix} S_1 & T_1 & & & & \times & \times \\ & & & & & \times & \times \\ & & & & & \times & \times \\ & S_2 & T_2 & & & \times & \times \\ & & \ddots & \ddots & & \vdots & \vdots \\ & & & & & \times & \times \\ & & & S_N & T_N & \times & \times \\ & & & & & \times & \times \\ D_a & & & & D_b & \times & \times \\ \times\times & \times\times & & \cdots & \times\times & O & O \\ \times\times & \times\times & & \cdots & \times\times & \times & \times \end{pmatrix}. \tag{2..29}$$

The last two rows of $\mathcal{A}$ arise from (2..26) and (2..27), and the last two columns correspond to X and λ. The package *auto* [52,53,54] for computing periodic orbits etc., solves a related problem. Its linear algebra involves coefficient matrices with a structure similar to (2..29).

2.4.4. Parameter Identification As in [29], we define a standard problem. Find $\mathbf{y}(x) \in \mathcal{R}^n$, $\boldsymbol{\lambda} \in \mathbf{R}^m$: (i) to minimize

$$\left\| \sum_{i=1}^{N} M_i \mathbf{y}(x_i) + Z\boldsymbol{\lambda} - \mathbf{d} \right\|_2^2, \tag{2..30}$$

where M_i, Z, $\mathbf{d}$ are given and problem dependent; and (ii) to satisfy the ODE system

$$\mathbf{y}'(x) = A(x)\mathbf{y}(x) + C(x)\boldsymbol{\lambda} + \mathbf{f}(x). \tag{2..31}$$

Essentially, (2..30)-(2..31) comprise an equality constrained linear least squares problem. Using any standard numerical technique to discretize the ODE (2..31), we obtain a discrete form

$$\min_{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_N, \boldsymbol{\lambda}} \left\| \sum_{i=0}^{N} M_i \mathbf{y}_i + Z\boldsymbol{\lambda} - \mathbf{d} \right\|_2^2 \tag{2..32}$$

subject to the constraints

$$S_i\mathbf{y}_i + T_i\mathbf{y}_{i+1} + F_i\boldsymbol{\lambda} = \mathbf{f}_i, \quad S_i, T_i \in \mathcal{R}^{n\times n}, \quad F_i \in \mathcal{R}^{m\times n}, \quad i = 1, 2, \ldots, N. \quad (2..33)$$

Rather than solve (2..32)-(2..33), Mattheij and S.J. Wright [117] impose extra side constraints

$$C_i\mathbf{y}_i + D_i\mathbf{y}_{i+1} + E_i\boldsymbol{\lambda} = \mathbf{g}_i, \quad i = 1, 2, \ldots, N. \quad (2..34)$$

Problem (2..32)-(2..34) gives rise to a linear system with coefficient matrix

$$\begin{pmatrix} S_1 & T_1 & & & & F_1 \\ & S_2 & T_2 & & & F_2 \\ & & \ddots & \ddots & & \vdots \\ & & & S_N & T_N & F_N \\ C_1 & D_1 & & & & E_1 \\ & C_2 & D_2 & & & E_2 \\ & & \ddots & \ddots & & \vdots \\ & & & C_N & D_N & E_N \end{pmatrix}.$$

To solve this system, a stable compactification scheme [117] which can take account of rank deficiency may be used. See Section 5.1.3. for a discussion of a cyclic reduction algorithm which can be modified to solve the system.

Another example where ABDs arise in parameter estimation, this time associated with differential-algebraic equations, is given in [1].

3. Partial Differential Equations and ABD Systems

OSC methods for partial differential equations (PDEs) are a rich source of ABD systems. Such systems arise in OSC methods for separable elliptic boundary value problems, and in the method of lines for various initial-boundary value problems. Also, ABD systems arise in Keller's box scheme for parabolic initial-boundary value problems [94].

3.1. OSC for Separable Elliptic Boundary Value Problems

Many fast direct methods exist for solving the linear systems arising in the numerical solution of separable elliptic PDEs posed in the unit square. An important class comprises matrix decomposition algorithms, which have been formulated for finite difference, finite element Galerkin, OSC and spectral methods [22]. To describe how ABD systems arise in OSC algorithms [23], consider the elliptic boundary value problem

$$(L_1+L_2)u = f(x_1, x_2), \quad (x_1, x_2) \in \Omega = (a, b)\times(a, b); \quad u(x_1, x_2) = 0, \quad (x_1, x_2) \in \partial\Omega, \quad (3..1)$$

where

$$L_i u = -a_i(x_i)\frac{\partial^2 u}{\partial x_i^2} + b_i(x_i)\frac{\partial u}{\partial x_i} + c_i(x_i)u, \quad i = 1, 2, \quad (3..2)$$

with $a_i > 0, c_i \geq 0,\ i = 1, 2$, and $b_1 = 0$.

Using the notation of Section 2.3., the OSC approximation $U(x_1, x_2) \in \mathcal{M}_r^0(\pi) \otimes \mathcal{M}_r^0(\pi)$ satisfies

$$(L_1 + L_2)U(\xi_{m_1}, \xi_{m_2}) = f(\xi_{m_1}, \xi_{m_2}), \qquad m_1, m_2 = 1, 2, \ldots, M, \tag{3..3}$$

where, as before, $M = N(r-1)$. Let $\{\phi_n\}_{n=1}^M$ be a basis for $\mathcal{M}_r^0(\pi)$. If

$$U(x_1, x_2) = \sum_{n_1=1}^{M} \sum_{n_2=1}^{M} u_{n_1,n_2} \phi_{n_1}(x_1)\phi_{n_2}(x_2),$$

with coefficients $\mathbf{u} = [u_{1,1}, u_{1,2}, \ldots, u_{1,M}, \ldots, u_{M,1}, u_{M,2}, \ldots, u_{M,M}]^T$, and with $\mathbf{f} = [f_{1,1}, f_{1,2}, \ldots, f_{1,M}, \ldots, f_{M,1}, f_{M,2}, \ldots, f_{M,M}]^T$, where $f_{m_1,m_2} = f(\xi_{m_1}, \xi_{m_2})$, the matrix-vector form of (3..3) is

$$(A_1 \otimes B_2 + B_1 \otimes A_2)\mathbf{u} = \mathbf{f}, \tag{3..4}$$

where $A_i = (L_i\phi_n(\xi_m))_{m,n=1}^M$, $B_i = (\phi_n(\xi_m))_{m,n=1}^M$, and $\otimes$ denotes the tensor (Kronecker) product. If the functions $\{\phi_n\}_{n=1}^M$ are Hermite type, B-splines, or monomial basis functions, these matrices are ABD, having the structure (2..5), (2..10) or (2..11), respectively.

Now, let $W = \text{diag}(h_1w_1, h_1w_2, \ldots, h_1w_{r-1}, \ldots, h_Nw_1, h_Nw_2, \ldots, h_Nw_{r-1})$ and, for v defined on $[a, b]$, $D(v) = \text{diag}(v(\xi_1), v(\xi_2), \ldots, v(\xi_M))$. If

$$F_1 = B_1^T W D(1/a_1) B_1, \qquad G_1 = B_1^T W D(1/a_1) A_1, \tag{3..5}$$

then F_1 is symmetric and positive definite, and G_1 is symmetric [23]. Hence, there exist real $\Lambda = \text{diag}(\lambda_j)_{j=1}^M$ and a real nonsingular Z such that

$$Z^T G_1 Z = \Lambda, \qquad Z^T F_1 Z = I. \tag{3..6}$$

By (3..5), Λ, Z can be computed using the decomposition $F_1 = LL^T$, $L = B_1^T[WD(1/a_1)]^{1/2}$, and solving the symmetric eigenproblem for $C = L^{-1}G_1L^{-T} = [WD(1/a_1)]^{1/2}A_1B_1^{-1}[WD(1/a_1)]^{-1/2}$,

$$Q^T C Q = \Lambda \tag{3..7}$$

with Q orthogonal. If $Z = B_1^{-1}[WD(1/a_1)]^{-1/2}Q$, then Λ, Z satisfy (3..6). Thus,

$$[Z^T B_1^T W D(1/a_1) \otimes I](A_1 \otimes B_2 + B_1 \otimes A_2)(Z \otimes I) = \Lambda \otimes B_2 + I \otimes A_2,$$

leading to the matrix decomposition algorithm in Algorithm 3.1 for solving (3..4), where steps 1, 3, and 4 each involve solving M independent ABD systems which are all of order M. In Step 1, C can be determined efficiently by solving $B_1^T[WD(1/a_1)]^{1/2}C = A_1^T[WD(1/a_1)]^{1/2}$. Computing the columns of C requires solving linear systems with coefficient matrix $\{[WD(1/a_1)]^{1/2}B_1\}^T$, the transpose of the ABD matrix in Step 4. The ABD matrix is factored once and the columns of C determined. This factored form is also used in Step 4. In Step 3, the ABD matrices have the form $A_2 + \lambda_j B_2$, $j = 1, 2, \ldots, M$; this step is equivalent to solving a system of BVODEs.

In [24], a matrix decomposition algorithm is formulated for solving the linear systems arising in OSC with piecewise Hermite bicubic basis functions on a uniform mesh applied to (3..1)-(3..2) with $a_1 = 1$ and $c_1 = 0$ (a similar method is discussed in [140]). The algorithm comprises steps 2-4 of Algorithm 3.1 with $W = \frac{h}{2}I$ and $D = I$ and uses *explicit*

	Algorithm 3.1
1.	Determine Λ and Q satisfying (3..7)
2.	Compute $\mathbf{g} = (Q^T[WD(1/a_1)]^{1/2} \otimes I_2)\mathbf{f}$
3.	Solve $(\Lambda \otimes B_2 + I_1 \otimes A_2)\mathbf{v} = \mathbf{g}$
4.	Compute $\mathbf{u} = (B_1^{-1}[W_1 D_1(1/a_1)]^{-1/2} Q \otimes I_2)\mathbf{v}$

formulas for Λ and Z for appropriately scaled basis functions. Extensions to Neumann, periodic and mixed BCs are discussed in [15,28,65,136], and to problems in three dimensions in [128]. Progress has also been made on extensions to OSC with higher degree piecewise polynomials, [139].

ABD systems also arise in alternating direction implicit (ADI) methods for the OSC equations (3..4); see [19,44,45,46,47,60]. For example, consider (3..1) with L_i given by (3..2) and $b_i = 0, i = 1, 2$. The ADI OSC method of [19] is: given $\mathbf{u}^{(0)}$, for $k = 0, 1, \ldots,$ compute $\mathbf{u}^{(k+1)}$ from

$$\begin{aligned} [(A_1 + \gamma_{k+1}^{(1)} B_1) \otimes B_2]\mathbf{u}^{(k+1/2)} &= \mathbf{f} - [B_1 \otimes (A_2 - \gamma_{k+1}^{(1)} B_2)]\mathbf{u}^{(k)}, \\ [B_1 \otimes (A_2 + \gamma_{k+1}^{(2)} B_2)]\mathbf{u}^{(k+1)} &= \mathbf{f} - [(A_1 - \gamma_{k+1}^{(2)} B_1) \otimes B_2]\mathbf{u}^{(k+1/2)}, \end{aligned} \tag{3..8}$$

where $\gamma_{k+1}^{(1)}$ and $\gamma_{k+1}^{(2)}$ are acceleration parameters. Using properties of the tensor product, it is easy to see that each step requires solving independent sets of ABD systems. For example, $[(A_1 + \gamma B_1) \otimes B_2]\mathbf{v} = \mathbf{g}$ is equivalent to $[(A_1 + \gamma B_1) \otimes I]\mathbf{w} = \mathbf{g}$, $(I \otimes B_2)\mathbf{v} = \mathbf{w}$. ABD systems also occur in certain block iterative methods for solving (3..4), see [137,138].

ABD systems arise in other methods for elliptic problems. For example, they appear in Fourier analysis cyclic reduction (FACR) OSC methods for the Dirichlet problem for Poisson's equation in the unit square [20], in fast direct OSC methods for biharmonic problems [21,108,109,135], and in spectral methods for the steady state PDEs of fluid mechanics [49,86,87,88].

3.2. OSC for Time Dependent Problems

Consider the parabolic initial boundary value problem

$$\begin{aligned} \frac{\partial u}{\partial t} + Lu &= f(x,t), \quad (x,t) \in (a,b) \times (0,T], \\ \alpha_a u(a,t) + \beta_a \frac{\partial u}{\partial x}(a,t) = g_a(t), &\qquad \alpha_b u(b,t) + \beta_b \frac{\partial u}{\partial x}(b,t) = g_b(t), \quad t \in (0,T], \\ u(x,0) &= u_0(x), \quad x \in (a,b), \end{aligned} \tag{3..9}$$

where

$$Lu = -a(x,t)\frac{\partial^2 u}{\partial x^2} + b(x,t)\frac{\partial u}{\partial x} + c(x,t)u.$$

In a method of lines approach, we construct the continuous-time OSC approximation which is a differentiable map $U : [0,T] \to \mathcal{M}_r(\pi)$ such that [58]

$$\alpha_a U(a,t) + \beta_a \frac{\partial U}{\partial x}(a,t) = g_a(t), \quad t \in (0,T],$$

$$\begin{aligned}\left[\frac{\partial U}{\partial t} + LU\right](\xi_i, t) &= f(\xi_i, t), \quad i = 1, 2, \ldots, M, \quad t \in (0, T], \quad (3..10)\\ \alpha_b U(b,t) + \beta_b \frac{\partial U}{\partial x}(b,t) &= g_b(t), \quad t \in (0,T],\\ U(\xi_i, 0) &= u_0(\xi_i), \qquad i = 1, 2, \ldots, M.\end{aligned}$$

With $U(x,t) = \sum_{j=1}^{M+2} u_j(t)\phi_j(x)$, where $\{\phi_j\}_{j=1}^{M+2}$ is a basis for $\mathcal{M}_r(\pi)$, (3..10) is an ODE initial value system

$$B\mathbf{u}'(t) + A(t)\mathbf{u}(t) = \mathbf{F}(t), \quad t \in (0,T], \qquad \mathbf{u}(0) \text{ prescribed}, \qquad (3..11)$$

where B and $A(t)$ are both ABD. An example of a discrete-time OSC method is the (second order in time) Crank-Nicolson collocation method for (3..10) which determines $\{U^k\}_{k=1}^{Q} \subset \mathcal{M}_r(\pi)$ satisfying the BCs such that

$$\left[\frac{U^{k+1} - U^k}{\Delta t} + L_{k+1/2}U^{k+1/2}\right](\xi_j) = f(\xi_j, t_{k+1/2}), \quad j = 1, 2, \ldots, M, \quad k = 0, 1, \ldots, Q-1,$$

where $U^{k+1/2} = (U^k + U^{k+1})/2$, $t_{k+1/2} = (k+1/2)\Delta t$, $L_{k+1/2} = -a(x, t_{k+1/2})\dfrac{\partial^2}{\partial x^2} + b(x, t_{k+1/2})\dfrac{\partial}{\partial x} + c(x, t_{k+1/2})$, and $Q\Delta t = T$. Essentially, this is the trapezoidal method for (3..11):

$$[B + \frac{1}{2}\Delta t A(t_{k+1/2})]\mathbf{u}^{k+1} = [B - \frac{1}{2}\Delta t A(t_{k+1/2})]\mathbf{u}^k + \Delta t \mathbf{F}(t_{k+1/2}). \qquad (3..12)$$

Thus, with a standard basis, an ABD system must be solved at each time step.

When the PDE in (3..9) has time independent coefficients and right hand side, the solution of (3..11) is $\mathbf{u}(t) = \exp(-tB^{-1}A)[\mathbf{u}(0) - A^{-1}\mathbf{F}] + A^{-1}\mathbf{F}$, or, in incremental form,

$$\mathbf{u}((k+1)\Delta t) = \exp(-\Delta t B^{-1}A)[\mathbf{u}(k\Delta t) - A^{-1}\mathbf{F}] + A^{-1}\mathbf{F}. \qquad (3..13)$$

One way to evaluate (3..13) is to use a Padé approximation to the matrix exponential. The (1,1) Padé approximation gives the Crank-Nicolson method in (3..12). A fourth order in time approximation can be obtained using the (2,2) Padé approximation. At each time step, this gives a linear system with a coefficient matrix which is quadratic in A and $\Delta t B$. On factoring this quadratic, we obtain a pair of complex ABD systems with complex conjugate coefficient matrices $B + \beta\Delta t A$ and $B + \bar{\beta}\Delta t A$, where $\beta = (1 + i\sqrt{3})/4$, [64]. However, a system with just one of these coefficient matrices must be solved, then the fact that equation (3..13) is real can be exploited to compute directly the (real) solution of the discretized system. See [77] for more on solving the linear systems arising from using higher order approximations to the matrix exponential in (3..13).

ABD linear systems similar to those in (3..12) arise in ADI OSC methods for parabolic and hyperbolic initial-boundary value problems in two space variables (see [25,26,27,70,71] and references cited in these papers), and in ADI OSC methods for Schrödinger problems [106] and vibration problems [107] also in two space variables. OSC methods with monomial basis functions for solving Schrödinger equations [130,131,132,133,134], the Kuramoto-Sivashinsky equation [116] and the Roseneau equation [115], all in one space variable, involve linear systems of the form (2..11).

3.3. Keller's Box Scheme

Following [94], we set $v = \partial u/\partial x$ and reformulate (3..9) as an initial-boundary value problem for a first order system of PDEs on $(a, b) \times (0, T]$,

$$\begin{aligned} \frac{\partial u}{\partial x} &= v, \\ a(x,t)\frac{\partial v}{\partial x} &= \frac{\partial u}{\partial t} + b(x,t)v + c(x,t)u - f(x,t), \\ \alpha_0 u(a,t) + \beta_0 v(a,t) = g_0(t), &\quad \alpha_1 u(b,t) + \beta_1 v(b,t) = g_1(t) \quad t \in (0,T], \\ u(x,0) &= u_0(x), \quad x \in (a,b). \end{aligned} \tag{3..14}$$

Using the mesh π, at each time step the box scheme applied to (3..14) gives an ABD system with coefficient matrix (2..5), where $S_i, T_i \in \mathcal{R}^{2\times 2}$.

Slightly different ABD systems arise in the solution of nonlocal parabolic problems modeling certain chemical and heat conduction processes [69]. As an example, consider

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, \quad (x,t) \in (a,b) \times (0,T], \\ \int_a^\eta u(s,t)ds = F(t), &\quad \frac{\partial u}{\partial x}(a,t) = g_0(t), \quad t \in (0,T], \\ u(x,0) &= u_0(x), \quad x \in (a,b), \end{aligned} \tag{3..15}$$

where $\eta \in (a, b)$ is given (cf. Section 2.4.3.). With $w(x,t) = \int_a^x u(s,t)ds$, $(x,t) \in (a,\eta)\times (0,T]$, (3..15) is written as a first order system as in (3..14) and the box scheme is applied to the reformulated problem. At each time step, the ABD coefficient matrix is again of the form (2..5), but now, if η is a grid-point, $\eta = x_K$, say, then $S_i \in \mathcal{R}^{3\times 3},\ i = 1, 2, \ldots, K$, and $S_i \in \mathcal{R}^{2\times 2}$ otherwise, $T_i \in \mathcal{R}^{3\times 3},\ i = 1, 2, \ldots, K-1$, $T_K \in \mathcal{R}^{3\times 2}$, $T_i \in \mathcal{R}^{2\times 2},\ i = K+1, K+2, \ldots, N$, and $D_a = [0\ 0\ 1]$, $D_b = [0\ 1]$.

In [68], a similar nonlocal parabolic problem was solved using OSC, again involving ABD systems.

4. Direct Sequential Solvers for ABD and BABD Systems

4.1. Solvers for ABD systems

We describe the essential features of the algorithms using a simple example with a coefficient matrix of the form in Figure 1..2, namely the matrix of Figure 4..1 in which there are two 4×7 blocks $W^{(1)}, W^{(2)}$, and *TOP* and *BOT* are 2×3 and 1×3, respectively. The overlap between successive blocks is thus 3.

4.1.1. Gaussian Elimination with Partial Pivoting The procedure implemented in *solveblok* [33,34] uses conventional Gaussian elimination with partial pivoting. Fill-in may be introduced in the positions indicated $*$ in Figure 4..2. The possible fill-in, and consequently the possible additional storage and work, depends on the number of rows, N_T, in the block *TOP*. Stability is guaranteed by standard results for banded systems [30].

4.1.2. Alternate Row and Column Elimination This stable elimination procedure, based on the approach of Varah [142], generates no fill-in for the matrix $\mathcal{A}$ of Figure 4..1.

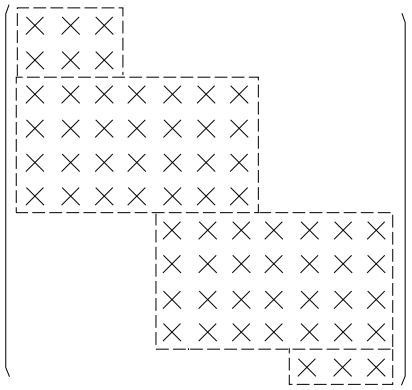

Figure 4..1. Structure of the example matrix

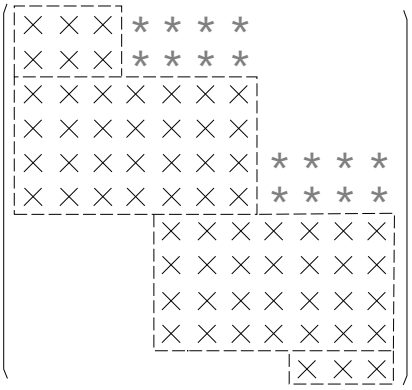

Figure 4..2. Fill-in introduced by *SOLVEBLOK*

Suppose we choose a pivot from the first row. If we interchange the first column and the column containing the pivot, there is no fill-in. Moreover, if instead of performing row elimination as in conventional Gaussian elimination, we reduce the $(1,2)$ and $(1,3)$ elements to zero by column elimination, the corresponding multipliers are bounded in magnitude by unity. We repeat this process in the second step, choosing a pivot from the elements in the $(2,2)$ and $(2,3)$ positions, interchanging columns 2 and 3 if necessary and eliminating the $(2,3)$ element. If this procedure were adopted in the third step, fill-in could be introduced in the $(i,3)$ positions, $i = 7,8,9,10$. To avoid this, we switch to row elimination with partial pivoting to eliminate the $(4,3)$, $(5,3)$, $(6,3)$ elements, which does not introduce fill-in. We continue using row elimination with partial pivoting until a step is reached when fill-in could occur, at which point we switch back to the "column pivoting, column elimination" scheme. This strategy leads to a decomposition

$$\mathcal{A} = PL\tilde{\mathcal{B}}UQ,$$

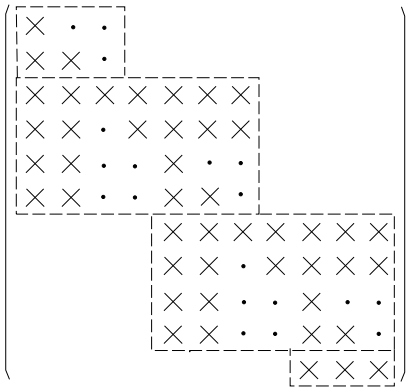

Figure 4..3. Structure of the reduced matrix

where P, Q are permutation matrices recording the row and column interchanges, respectively, the unit lower and unit upper triangular matrices L, U contain the multipliers used in the row and column eliminations, respectively, and the matrix $\tilde{\mathcal{B}}$ has the structure shown in Figure 4..3, where $\cdot$ denotes a zeroed element. Since there is only one row or column interchange at each step, the pivotal information can be stored in a single vector of the order of the matrix, as in conventional Gaussian elimination. The nonzero elements of L, U can be stored in the zeroed positions in $\mathcal{A}$. The pattern of row and column eliminations is determined by N_T and the number of rows N_W in a general block $W^{(i)}$ (cf. Figure 1..2). In general, a sequence of N_T column eliminations is alternated with a sequence of $N_W - N_T$ row eliminations (see [50] for details). For a further analysis of this and related approaches, see [129].

To solve $\mathcal{A}\mathbf{x} = \mathbf{b}$, we solve

$$PL\mathbf{z} = \mathbf{b}, \qquad \tilde{\mathcal{B}}\mathbf{w} = \mathbf{z}, \qquad UQ\mathbf{x} = \mathbf{w}.$$

The second step requires particular attention. If the components of $\mathbf{w}$ are ordered so that those associated with the column eliminations precede those associated with the row eliminations, and the equations are ordered accordingly, the system is reducible. In our example, if we use the ordering

$$\hat{\mathbf{w}} = [w_1, w_2, w_5, w_6, w_9, w_{10}, w_3, w_4, w_7, w_8, w_{11}]^T,$$

the coefficient matrix of the reordered equations has the structure in Figure 4..4. Thus, the components $w_1, w_2, w_5, w_6, w_9, w_{10}$, are determined by solving a lower triangular system and the remaining components by solving an upper triangular system.

4.1.3. Modified Alternate Row and Column Elimination The alternate row and column elimination procedure can be made more efficient by using the fact that, after each sequence of row or column eliminations, a reducible matrix results. This leads to a reduction in the number of arithmetic operations because some operations involving matrix-matrix multiplications in the decomposition phase can be deferred to the solution phase, where only matrix-vector multiplications are required. After the first sequence of column eliminations, involving a permutation matrix Q_1 and multiplier matrix U_1, say, the resulting

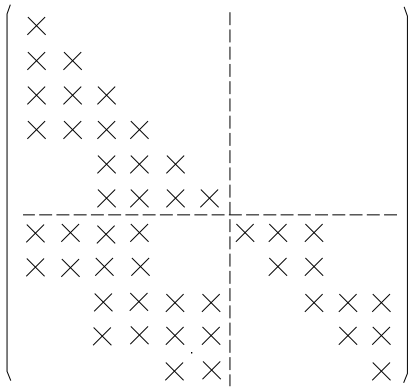

Figure 4..4. The coefficient matrix of the reordered system

matrix is

$$\mathcal{B}_1 = \mathcal{A}Q_1U_1 = \left(\begin{array}{c|c} C_1 & O \\ \hline M_1 & \\ & \mathcal{A}_1 \\ O & \end{array}\right),$$

where $C_1 \in \mathcal{R}^{2\times 2}$ is lower triangular, $M_1 \in \mathcal{R}^{4\times 2}$ and $\mathcal{A}_1 \in \mathcal{R}^{9\times 9}$. The equation $\mathcal{A}\mathbf{x} = \mathbf{b}$ becomes

$$\mathcal{B}_1\hat{\mathbf{x}} = \mathbf{b}, \quad \hat{\mathbf{x}} = U_1^{-1}Q_1^T\mathbf{x} = \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \end{pmatrix},$$

and $\hat{\mathbf{x}}_1 \in \mathcal{R}^2$. Setting $\mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$, where $\mathbf{b}_1 \in \mathcal{R}^2$, we obtain

$$C_1\hat{\mathbf{x}}_1 = \mathbf{b}_1, \quad \mathcal{A}_1\hat{\mathbf{x}}_2 = \mathbf{b}_2 - \begin{pmatrix} M_1\hat{\mathbf{x}}_1 \\ \mathbf{0} \end{pmatrix} \equiv \hat{\mathbf{b}}_2. \tag{4..1}$$

The next sequence of eliminations, that is, the first sequence of row eliminations, is applied only to $\mathcal{A}_1$ to produce another reducible matrix

$$L_1P_1\mathcal{A}_1 = \left(\begin{array}{c|cc} R_1 & N_1 & O \\ \hline O & \multicolumn{2}{c}{\mathcal{A}_2} \end{array}\right),$$

where $R_1 \in \mathcal{R}^{2\times 2}$ is upper triangular, $N_1 \in \mathcal{R}^{2\times 3}$ and $\mathcal{A}_2 \in \mathcal{R}^{7\times 7}$. If

$$\hat{\mathbf{x}}_2 = \begin{pmatrix} \tilde{\mathbf{x}}_1 \\ \tilde{\mathbf{x}}_2 \end{pmatrix}, \qquad L_1P_1\hat{\mathbf{b}}_2 = \begin{pmatrix} \tilde{\mathbf{b}}_1 \\ \tilde{\mathbf{b}}_2 \end{pmatrix},$$

where $\tilde{\mathbf{x}}_1,\ \tilde{\mathbf{b}}_1 \in \mathcal{R}^2$, system (4..1) becomes

$$\mathcal{A}_2\tilde{\mathbf{x}}_2 = \tilde{\mathbf{b}}_2, \quad R_1\tilde{\mathbf{x}}_1 = \tilde{\mathbf{b}}_1 - [N_1 \quad O]\,\tilde{\mathbf{x}}_2, \tag{4..2}$$

and the next sequence of eliminations is applied to $\mathcal{A}_2$, which has the structure of the original matrix with one W block removed. Since row operations are not performed on

M_1 in the second elimination step, and column operations are not performed on N_1 in the third, etc., there are savings in arithmetic operations [50]. The decomposition phase differs from that in alternating row and column elimination in that if the r^{th} elimination step is a column (row) elimination, it leaves unaltered the first $(r-1)$ rows (columns) of the matrix. The matrix in the modified procedure has the same structure and diagonal blocks as $\tilde{B}$, and the permutation and multiplier matrices are identical. This procedure is Gaussian elimination with a special form of partial pivoting [129].

In [50,51], this modified alternate row and column elimination procedure was developed and implemented in the package *colrow* for systems with matrices of the form in Figure 1..2, and in the package *arceco* for ABD systems in which the blocks are of varying dimensions, and the first and last blocks protrude, as shown in Figure 1..1. The latter package has been used to solve the systems arising in Keller's box scheme applied to (3..15), [69].

Numerical experiments [75] demonstrate the effectiveness of *colrow* and *arceco* and their superiority over *solveblok* on systems arising from BVODEs. The *arceco* package was modified and augmented in [35] to use the level 2 *BLAS* [56], and the code *f01lhf* included in the NAG Library. A level 3 *BLAS* [55] version of *arceco* was developed in [76,127], and a corresponding code [48] carefully avoids the implicit fill-in due to blocking in the *BLAS*.

Two variants of *colrow* have been produced by Keast [89,90]. Like *f01lhf*, the first [89] solves not only an ABD system but also a system whose coefficient matrix is the transpose of an ABD matrix. These codes may be employed in Steps 1 and 4 of Algorithm 3.1. The second [90] solves complex ABD systems and may be employed in the Padé methods of Section 3.4.

4.1.4. Lam's Method In Lam's method [102], alternate row and column interchanges are used to avoid fill-in but only row elimination is performed throughout. This leads to a $PLUQ$ decomposition in which the elements of L, the multipliers in the elimination process, may not be bounded *a priori*. This technique, which motivated Varah's algorithm [142], was rediscovered in an application arising in the analysis of beam structures [17]; its numerical stability is analyzed in [141]. The algorithm is implemented in *lampak* [91]. It gives essentially the same results as *colrow* [50,51] in all numerical experiments known to its authors.

In [75,82], an early attempt to compare the performance of the various codes on "supercomputers" examines the performance of a vectorized *solveblok, colrow, arceco* and *lampak* on a CDC Cyber 205. The vectorized versions of *colrow, arceco* and *lampak* perform about equally and significantly more efficiently than the redesigned *solveblok* on a wide range of test problems. However, the nature of the memory hierarchy on the CDC Cyber 205 was very different from that of modern vector machines, so these conclusions may no longer be valid.

4.1.5. Special ABD Systems When modified alternate row and column elimination is used to solve ABD systems arising in multiple shooting or the condensed OSC equations (2..13) for BVODEs with separated BCs, no advantage is taken of the sparsity of the "upper block diagonal" identity matrices. Hence the procedure introduces fill-in in these blocks as can be easily seen by considering the structure in Figure 4..5; the pattern of eliminations is one column elimination followed by two row eliminations. A simple change minimizes the fill-in and ensures that the reduced matrix has the same structure as would result if no pivoting at all were performed. Referring to Figure 1..2, the required modification is: if, at a row elimination step, we must interchange rows k and l of the current block $W^{(i)}$ then, before the row elimination, also interchange columns k and l of the submatrix of

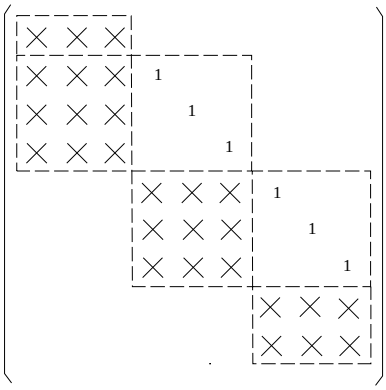

Figure 4..5. Matrix arising in multiple shooting

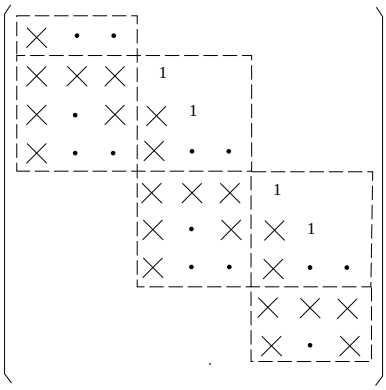

Figure 4..6. Structure of the reduced matrix

$W^{(i)}$ originally containing the unit matrix and interchange columns k and l of the next block $W^{(i+1)}$. This decomposition yields the reduced (reducible) matrix in Figure 4..6. A package, *abbpack*, for solving this type of ABD linear system has been developed by Majaess et al., [111,112,113,114].

ABD linear systems of the form (2..11) cannot be solved using *colrow* or *arceco* [50,51] without fill-in. For such ABD systems, *abdpack* implements an alternate row and column elimination algorithm which, to avoid unnecessary fill-in, exploits the sparsity of the identity matrix, as described above, when rows of the current block D_i are interchanged in a row elimination step. Numerical experiments reported in [113,114] demonstrate *abdpack*'s superiority over the version of *solveblok* used in *colnew* [14]. ABD systems where the blocks overlap in rows were discussed in [67,92] in the context of finite element methods, and a modified alternate row and column elimination scheme for such systems is implemented in *rowcol* [66].

4.1.6. ABD Solvers in Packages The ABD packages described above are employed in packaged software for solving systems of nonlinear BVODEs and for solving systems of nonlinear initial-boundary value problems in one space variable using a method of lines approach with OSC in the spatial variable. The BVODE package *colsys* [9] uses *solveblok* [33,34] to solve linear systems of the form (2..10) arising when using OSC at Gauss points with B-spline bases. The packages *colnew* [14] and *pcolnew* [16] employ modified versions of *solveblok*, which implement essentially the same algorithm, to solve the ABD systems (2..12) resulting from condensation of the larger linear systems arising when using OSC at Gauss points with monomial spline bases. The NAG Library's simplified interface version of *colnew*, subroutine *d02tkf* due to Brankin and Gladwell, uses the NAG ABD code *f01lhf* [35]. The code *colrow* [50,51] is used in the MIRK BVODE code *mirkdc* [62], and a modified version of *colrow* is used in the deferred correction code *twpbvp* [42]. A modified version of *colnew*, *colmod* [38,146], uses a revised mesh selection strategy, automatic continuation and the modified *colrow*. The code *acdc* [39] uses automatic continuation, OSC at Lobatto points and the modified *colrow* for singularly perturbed BVODEs. Software implementing the SLU algorithm [147] for a number of different shared memory computers was developed by Remington [15] who investigated its use in the development of a parallel version of *colnew*. This software was also used for the parallel factorization and solution of ABD systems in an early version of *mirkdc* [121].

For PDEs, the code *pdecol* [110] uses *solveblok* [33,34] to solve the linear systems of the form (2..10) arising from using B-spline bases. The code *epdcol* [93] is a variant of *pdecol* in which *solveblok* is replaced by *colrow* [50,51]. In the method of lines code based on OSC with monomial bases described in [122], the linear systems are solved using *abdpack* [114].

4.2. Solvers for BABD Systems

In the context of structure, it is natural to consider Gaussian elimination with row interchanges as an appropriate method for BABD systems (5..1). We show why this approach can be unstable and we suggest some alternative approaches which may be used to exploit available software for ABD systems for the BABD case.

4.2.1. A Cautionary Example S.J. Wright [148] showed that conventional Gaussian elimination with row interchanges can be unstable for BABD systems (5..1) arising from multiple shooting. By extension, the BABD systems arising in finite difference and OSC discretizations are potentially unstable. Consider the linear ODE as of (2..2) but with nonseparated BCs (2..16) and

$$A(x) = \tilde{A} = \begin{pmatrix} -\frac{1}{6} & 1 \\ 1 & -\frac{1}{6} \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \ \ B_a = B_b = I, \ x \in [0, 60], \qquad (4..3)$$

which is well-conditioned [11]. Using any standard discretization leads to a linear system with matrix (5..1). Here, $T_i = T$, $S_i = S$ are constant and the resulting system should also be well-conditioned. Moving the BCs so that they become the first block equations and

premultiplying the resulting matrix $\mathcal{A}$ by $\mathcal{D} = \mathrm{diag}(I, T^{-1}, T^{-1}, \ldots, T^{-1})$ gives

$$\mathcal{D}\mathcal{A} = \bar{\mathcal{A}} = \begin{pmatrix} I & & & & & I \\ -B & I & & & & \\ & -B & I & & & \\ & & \ddots & \ddots & & \\ & & & -B & I & \end{pmatrix},$$

where $B = -T^{-1}S$. Suppose we are using the trapezoidal method, then, for h sufficiently small, $B = (I - h\tilde{A}/2)^{-1}(I + h\tilde{A}/2) \approx e^{h\tilde{A}}$, and all elements of B are less than one in magnitude. Using Gaussian elimination with partial pivoting, no interchanges are required, and

$$\bar{\mathcal{A}} = \mathcal{L}\mathcal{U} = \begin{pmatrix} I & & & & \\ -B & I & & & \\ & -B & I & & \\ & & \ddots & \ddots & \\ & & & -B & \hat{L} \end{pmatrix} \begin{pmatrix} I & & & I \\ & I & & B \\ & & \ddots & \vdots \\ & & I & B^{N-1} \\ & & & \hat{U} \end{pmatrix}.$$

The elements in the last column of $\mathcal{U}$ grow exponentially with N. This instability occurs when the BABD matrix $\tilde{A}$ has negative diagonal elements but is not restricted to this case nor to BVODEs with constant coefficient matrices. A similar analysis applies to the structured LU factorization in [149], denoted LU below; this algorithm is used in the comparisons in [99].

4.2.2. Sequential Methods for BABDs To our knowledge, there exists no sequential algorithm designed specifically for BABD systems; we describe a number of parallel algorithms in the next section. However, it is possible to use software described in [80] for general bordered systems. This software assumes a matrix of the form

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{4..4}$$

with A and D square and of orders n' and m', respectively, where m' is "small". It is also assumed that software is available which will solve linear systems of both the forms $A\mathbf{x} = \mathbf{b}$ and $A^T\mathbf{x} = \mathbf{b}$.

For BABD coefficient matrices of the form (2..15), that is with nonseparated BCs, we have $D = B_b$ and $n' = Nn$ and $m' = n$. In this case, A is block upper triangular. (Both B and C have some structure which cannot be directly exploited by the software described in [80].) In the case of partially separated BCs of the form (2..18) with matrix structure as in (2..19), the matrix A is ABD unless the separated BCs are lumped with the nonseparated BCs to give a structure of the form (2..15). In this case, the matrix A is block upper triangular.

The algorithm implemented in the software *bemw* [80] is an extension of that described in [79] for $m' = 1$. It determines in sequence the solutions of each of a set of linear systems with ABD coefficient matrices A in (4.3) and A^T. (Software is available for solving ABD linear systems with matrices A and A^T, for example, *f01lhf* [35] and *transcolrow* [89]. Software for solving block upper triangular systems is easily constructed using the level 3 *BLAS* [55].) The right hand side of each linear system involves a column in the border of the

BABD and the solutions of the previously solved systems. In the nonseparated case, there are n of these columns. In the partially separated case, the number of these columns is n minus the number of separated boundary conditions. A clever organization of the algorithm avoids using recursion. Stability is not guaranteed but there is empirical evidence that it is usually attained.

A related approach is described in [151]. There, block elimination is performed on the matrix (4..4). Small perturbations are introduced in the factorized matrices to avoid overflows, hence leading to inaccuracies in the solution. Then, iterative refinement is used to improve the solution. An extensive error analysis in [151] is somewhat inconclusive.

Numerical tests of this algorithm and of that discussed in [80] have been carried out on an SGI 8000, an SGI 10000 and a Cray J-9x; these tests are for the matrix A of dense and tridiagonal forms but not for A of ABD or block upper triangular form. Generally, for dense systems, the errors reported favor the new algorithm, whereas there is little difference between the algorithms for tridiagonal systems. The timings for the new algorithm are vastly superior but do depend on implementation features associated with using the level 3 *BLAS* [55]. It is reported that usually only one step of the iterative refinement algorithm is needed to compute near to exact solutions.

An alternative, guaranteed stable approach is to write the BABD system as an ABD system of twice the size. This is achieved by introducing dummy variables to "separate" the boundary conditions (see Section 2.4.1.). Then, use an ABD algorithm is used for the resulting ABD system which has internal blocks of size $2n$, ignoring structure internal to the blocks. The arithmetic cost is about eight times that of the factorization in *bemw*.

5. Direct Parallel Solvers for BABD and ABD Linear Systems

We consider algorithms for solving BABD and ABD linear systems in a parallel environment. We restrict attention to the systems arising when solving linear BVODEs where the ODE is as in (2..2) for both nonseparated and separated BCs. All the algorithms can be implemented efficiently to exploit medium granularity parallelism (i.e., where the number of processors, p, does not exceed the number of block rows of the BABD matrix). On each block row, we apply the same general decomposition. On a distributed memory machine, this corresponds to partitioning the problem by block rows among the processors.

5.1. Nonseparated BCs - Parallel Algorithms

For the mesh π, the linear BABD system obtained using any of the basic methods to discretize the linear ODE of (2..2) with BCs (2..16) has the structure:

$$\mathcal{A}\mathbf{x} \equiv \begin{pmatrix} S_1 & T_1 & & & \\ & S_2 & T_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & T_N \\ B_a & & & & B_b \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \\ \mathbf{c} \end{pmatrix} \equiv \mathbf{b}, \qquad (5..1)$$

where $S_i, T_i \in \mathcal{R}^{n\times n}$, $\mathbf{x}_i, \mathbf{f}_i, \mathbf{c} \in \mathcal{R}^n$. Assume $N = kp$. In all our partitioning algorithms (except wrap-around partitioning), the block row in (5..1) containing the BCs is temporarily neglected while the remaining block rows are shared among the processors. The i^{th}

processor, $i = 1, 2, \ldots, p$, stores the rectangular blocks of $\mathcal{A}$ in (5..1) associated with the unknowns $\mathbf{x}_{(i-1)k}, \mathbf{x}_{(i-1)k+1}, \ldots, \mathbf{x}_{ik}$,

$$\begin{pmatrix} S_{(i-1)k+1} & T_{(i-1)k+1} & & \\ & S_{(i-1)k+2} & T_{(i-1)k+2} & \\ & \ddots & \ddots & \\ & & S_{ik} & T_{ik} \end{pmatrix}. \tag{5..2}$$

Using different factorizations, each of the algorithms condenses the system to obtain an equation for the first and last unknowns corresponding to this block:

$$V_{ik}\mathbf{x}_{(i-1)k} + R_{ik}\mathbf{x}_{ik} = \mathbf{g}_{ik}. \tag{5..3}$$

Combining equations (5..3), $i = 1, 2, \ldots, p$, and the BCs, we obtain a reduced system with structure (5..1) and with unknowns $\mathbf{x}_0, \mathbf{x}_k, \ldots, \mathbf{x}_{pk}$. The process may be repeated cyclically (that is, recursively) using the same factorization successively on $p/2, p/4, p/8, \ldots$, processors, until a suitably small system is obtained. Then one processor is used to compute the direct factorization of the coefficient matrix of this system. An efficient strategy is to resort to direct factorization when sequential factorization at the current system size is more efficient than further recursion followed by a sequential factorization of a resulting smaller matrix, cf. [15,121].

An alternate approach for distributed memory machines always uses all p processors performing the same operations [3]. All the processors perform each reduction step; that is, there is redundant computation on otherwise idle processors. Hence, since quantities which would otherwise need to be communicated to processors are being computed where they are needed, the number of communications in the linear system solution phase may be reduced, actually by a half.

5.1.1. Local LU Factorization In the structured LU factorization [149] and in the stable local factorization (SLF-LU) partitioning algorithm [84], $(k-1)$ LU factorizations are performed in each block (5..2) on the $2n \times n$ sub-blocks corresponding to

$$\begin{pmatrix} T_{(i-1)k+j} \\ S_{(i-1)k+j+1} \end{pmatrix}.$$

Consider the first block $i = 1$ in (5..2). The factorization of the first sub-block is

$$\begin{pmatrix} T_1 \\ S_2 \end{pmatrix} = P_1 \begin{pmatrix} L_1 \\ \hat{S}_2 \end{pmatrix} U_1,$$

where $P_1 \in \mathcal{R}^{2n\times 2n}$ is a permutation matrix and $L_1, U_1 \in \mathcal{R}^{n\times n}$ are lower and upper triangular blocks, respectively. (The matrices P_j here and throughout this section arise from a partial pivoting strategy.) Thus,

$$\begin{pmatrix} S_1 & T_1 & \\ & S_2 & T_2 \end{pmatrix} = P_1 \begin{pmatrix} L_1 & \\ \hat{S}_2 & I \end{pmatrix} \begin{pmatrix} \hat{V}_1 & U_1 & \hat{T}_1 \\ V_2 & & R_2 \end{pmatrix}.$$

Continuing in this way, the j^{th} LU factorization, $j \geq 2$, computes

$$\begin{pmatrix} V_j & R_j & \\ & S_{j+1} & T_{j+1} \end{pmatrix} = P_j \begin{pmatrix} L_j & \\ \hat{S}_{j+1} & I \end{pmatrix} \begin{pmatrix} \hat{V}_j & U_j & \hat{T}_j \\ V_{j+1} & & R_{j+1} \end{pmatrix},$$

for rows j, $j+1$. After $k-1$ sub-block factorizations, the overall factorization of (5..2) for $i=1$ is

$$\mathcal{P}\begin{pmatrix} L_1 & & & & \\ \hat{S}_2 & L_2 & & & \\ & \hat{S}_3 & \ddots & & \\ & & \ddots & L_{k-1} & \\ & & & \hat{S}_{k-1} & I \end{pmatrix}\begin{pmatrix} \hat{V}_1 & U_1 & \hat{T}_1 & & & \\ \hat{V}_2 & & U_2 & \hat{T}_2 & & \\ \vdots & & & \ddots & \ddots & \\ \hat{V}_{k-1} & & & & U_{k-1} & \hat{T}_{k-1} \\ V_k & & & & & R_k \end{pmatrix},$$

where $\mathcal{P} \in \mathcal{R}^{kn\times kn}$ is a permutation matrix, the product of the $2n \times 2n$ permutation matrices P_j, $j = k-1, k-2, \ldots, 1$, expanded to size $kn \times kn$. Note that fill-in may occur, represented by the matrices $\hat{S}_j$, $\hat{T}_j$. For examples of this factorization in software for solving BVODEs, see [15,121].

In subsequent sections, we concentrate on factoring the first block of (5..2) as the process demonstrated there is simple to generalize to other blocks.

5.1.2. Local QR Factorization Note that here and in the following subsections we use the names of the blocks in the factorizations generically; the blocks are not the same as those with the same names in the local LU factorization. Using the QR factorization, the j^{th} step is

$$\begin{pmatrix} V_j & R_j & \\ & S_{j+1} & T_{j+1} \end{pmatrix} = Q_j \begin{pmatrix} \hat{V}_j & U_j & \hat{T}_j \\ V_{j+1} & & R_{j+1} \end{pmatrix},$$

where $Q_j \in \mathcal{R}^{2n\times 2n}$ is orthogonal, U_j is upper triangular and the other blocks are full. The complete factorization may be expressed in the form

$$\mathcal{Q}\begin{pmatrix} \hat{V}_1 & U_1 & \hat{T}_1 & & & \\ \hat{V}_2 & & U_2 & \hat{T}_2 & & \\ \vdots & & & \ddots & \ddots & \\ \hat{V}_{k-1} & & & & U_{k-1} & \hat{T}_{k-1} \\ V_k & & & & & R_k \end{pmatrix},$$

where $\mathcal{Q} \in \mathcal{R}^{kn\times kn}$ is orthogonal, the product of expanded (to size $kn \times kn$) versions of Q_j, $j = k-1, k-2, \ldots, 1$.

This algorithm is essentially equivalent to structured QR factorization [147] and SLF-QR partitioning [84]. It is stable in cases where LU factorization fails but costs twice as much as LU factorization when LU succeeds.

5.1.3. Local LU-CR Factorization Jackson and Pancer [84] and, independently, K. Wright [145] proposed combining cyclic reduction with LU factorization. Assuming k is a power of 2, for $j = 1, 3, \ldots, k-1$, the factorization is

$$\begin{pmatrix} S_j & T_j & \\ & S_{j+1} & T_{j+1} \end{pmatrix} = P_j \begin{pmatrix} L_j & \\ \hat{S}_{j+1} & I \end{pmatrix}\begin{pmatrix} V_j & U_j & \hat{T}_j \\ V_{j+1} & & R_{j+1} \end{pmatrix},$$

where the blocks have the structure and dimension of the corresponding blocks in Section 5.1.1.. In contrast to the earlier approaches, this involves $k/2$ simultaneous factorizations which can be distributed across $k/2$ processors.

Using odd-even permutation matrices $\bar{\mathcal{P}}^T, \hat{\mathcal{P}}$, the factorization can be recast as

$$\mathcal{P}\bar{\mathcal{P}}^T\mathcal{L}\left(\begin{array}{cccc|cccc} U_1 & & & & V_1 & \hat{T}_1 & & \\ & U_3 & & & & V_3 & \hat{T}_3 & \\ & & \ddots & & & & \ddots & \ddots \\ & & & U_{k-1} & & & & V_{k-1} \\ \hline & & & & V_2 & R_2 & & \hat{T}_{k-1} \\ & & & & & V_4 & R_4 & \\ & & & & & & \ddots & \ddots \\ & & & & & & & V_k \end{array}\right)\hat{\mathcal{P}}, \qquad R_k \tag{5..4}$$

where

$$\mathcal{L} = \left(\begin{array}{cccc|cccc} L_1 & & & & & & & \\ & L_3 & & & & & & \\ & & \ddots & & & & & \\ & & & L_{k-1} & & & & \\ \hline \hat{S}_2 & & & & I & & & \\ & \hat{S}_4 & & & & I & & \\ & & \ddots & & & & \ddots & \\ & & & \hat{S}_k & & & & I \end{array}\right)$$

is lower triangular and $\mathcal{P}$ is the product of the $P_j,\ j = 1, 3, \ldots, k-1$. The same approach may be applied to the submatrix

$$\left(\begin{array}{cccc} V_2 & R_2 & & \\ & V_4 & R_4 & \\ & & \ddots & \ddots \\ & & & V_k \end{array}\right) \qquad R_k \tag{5..5}$$

which has structure (5..2) and relates only the even unknowns $\mathbf{x}_0, \mathbf{x}_2, \ldots, \mathbf{x}_k$. (Set $S_j = V_{2j}, T_j = R_{2j}, j = 1, 2, \ldots, k/2$ and use factorization (5..4) with k replaced by $k/2$.) From this second step results a structure like (5..5) but of half the size, relating the unknowns $\mathbf{x}_0, \mathbf{x}_4, \ldots, \mathbf{x}_k$. After $\log_2 k$ steps of this recursion, there results a 2×2 block system for the unknowns $\mathbf{x}_0$, $\mathbf{x}_k$. After solving for these variables the recursion may be used in reverse to recover the others. As in the earlier algorithms, the recursion may be terminated before reaching the final 2×2 block system if it would be more efficient to solve directly a larger system than to proceed further recursively.

5.1.4. Local CR Factorization In [8], the local CR algorithm is proposed applying cyclic reduction directly to the block (5..2). In the first reduction step, for j odd,

$$\left(\begin{array}{ccc} S_j & T_j & \\ & S_{j+1} & T_{j+1} \end{array}\right) = \left(\begin{array}{cc} I & \\ \hat{S}_{j+1} & I \end{array}\right)\left(\begin{array}{ccc} S_j & T_j & \\ V_{j+1} & & T_{j+1} \end{array}\right), \tag{5..6}$$

where $\hat{S}_{j+1} = S_{j+1}T_j^{-1}$ and $V_{j+1} = -\hat{S}_{j+1}S_j$. The reduced matrix is (5..5) with $R_j = T_j$, which relates only the even unknowns. In (5..6) it is observed that this algorithm is unstable for several typical BVODEs because T_j can be ill-conditioned, hence (5..5) is potentially ill-conditioned even when (5..2) is well-conditioned.

In [7], a local centered CR factorization (CCR) is proposed to maintain stability and reduce computational cost. Consider the partitioning

$$\begin{pmatrix} S_j & T_j & & \\ & S_{j+1} & T_{j+1} & \end{pmatrix} = \begin{pmatrix} S_{j,1} & T_{j,1} & & \\ S_{j,2} & T_{j,2} & & \\ & S_{j+1,1} & T_{j+1,1} & \\ & S_{j+1,2} & T_{j+1,2} & \end{pmatrix}, \tag{5..7}$$

then $V_j = \begin{pmatrix} T_{j,2} \\ S_{j+1,1} \end{pmatrix}$ is a nonsingular square block of the size of S_j and T_j. Here, $S_{j,1}$, $S_{j,2}$, $T_{j,1}$, $T_{j,2}$, $j = 1, 2, \ldots, k-1$, are rectangular with n columns and with a number of rows in the range 0 to n. (In [7], attempts are made to choose this number to maintain stability.) The factorization for (5..7) is

$$P_j^T \begin{pmatrix} I_n & \\ \hat{S}_{j+1} & I_n \end{pmatrix} \left(\begin{array}{c|cc} V_j & S_{j,2} & \\ & & T_{j+1,1} \\ \hline & V_{j+1} & R_{j+1} \end{array}\right) P_j, \tag{5..8}$$

where $P_j \in \mathcal{R}^{2n \times 2n}$ is a permutation matrix,

$$\hat{S}_{j+1} = \begin{pmatrix} T_{j,1} \\ S_{j+1,2} \end{pmatrix} V_j^{-1}, \quad \begin{pmatrix} V_{j+1} & R_{j+1} \end{pmatrix} = \begin{pmatrix} S_{j,1} & \\ & T_{j+1,2} \end{pmatrix} - \hat{S}_{j+1} \begin{pmatrix} S_{j,2} & \\ & T_{j+1,1} \end{pmatrix}.$$

Of course V_j^{-1} is not calculated directly, rather the system is solved for $\hat{S}_{j+1}$. From the second of these equations, we obtain a block matrix (5..5) to which the procedure is applied recursively. This factorization is the local CR algorithm if $S_{j+1,1}$ is a null block. For the examples tested in [7], this algorithm is stable in cases where the blocks $T_{j,2}$ and $S_{j+1,1}$ are of equal size, for problems with the same number of BCs at each endpoint.

5.1.5. Local Stable CR Factorization Let

$$\begin{pmatrix} T_j \\ S_{j+1} \end{pmatrix} = P_j \begin{pmatrix} L_j \\ \hat{S}_{j+1} \end{pmatrix} U_j \tag{5..9}$$

be the factorization of the j^{th} column of (5..2). Starting from P_j in (5..9), build a permutation matrix $\bar{P}_j \in \mathcal{R}^{2n \times 2n}$ such that

$$\bar{P}_j \begin{pmatrix} T_j \\ S_{j+1} \end{pmatrix} = \begin{pmatrix} T_{j,1} \\ T_{j,2} \\ S_{j+1,1} \\ S_{j+1,2} \end{pmatrix},$$

where the rows $T_{j,2}$, $S_{j+1,1}$ contain the pivot elements of T_j, S_{j+1}, respectively. Then, there is a permutation matrix $\hat{P}_j \in \mathcal{R}^{n \times n}$ such that

$$V_j \equiv \begin{pmatrix} T_{j,2} \\ S_{j+1,1} \end{pmatrix} = \hat{P}_j L_j U_j. \tag{5..10}$$

where L_j, U_j are as in (5..9). By factoring as in (5..10), the algorithm proceeds as before. This algorithm has the stability properties of Gaussian elimination with partial pivoting [7].

5.1.6. RSCALE - a Scaled LU Approach Pancer and Jackson [84] give a set of comparisons of parallel algorithms for BABDs closely related to the LU and QR algorithms described earlier. In particular, they describe a new, rather complicated parallel algorithm, RSCALE. The main difference to the LU approach is that RSCALE introduces a scaling which ensures numerical stability for a wider class of BABD systems, for details see [84]. Overall in the tests, RSCALE delivers a similar accuracy to QR. but only involves about a half of the computational cost with a slightly lower memory requirement. The algorithm RSCALE has been employed in a new parallel version of *mirkdc* [85].

5.1.7. The Wrap-around Partitioning Algorithm Following the work in [59], Hegland and Osborne [83] recently described a wrap-around partitioning algorithm for BABD linear systems. An aim of the algorithm is to compute with long vectors. The algorithm transforms the BABD matrix by partitioning it into blocks that correspond to variables which can be eliminated independently of the others. After a local QR factorization (chosen on the grounds of stability) similar to that in the local QR factorization, a reduced BABD system results. The algorithm proceeds recursively. In [83], it is shown that small block sizes give best performance but that the optimal size depends on the computing system. Implementation and testing on a Fujitsu VPP500 vector computer are discussed. See [83] for details.

5.1.8. Computational Considerations - Parallel BABD Algorithms For simplicity, consider system (5..1) with N internal blocks on p processors, where $N = kp$, and n is the size of each internal block (equal to the number of first order ODEs). In the following, we use the acronyms: LU $\rightarrow$ local LU factorization; QR $\rightarrow$ local QR factorization; LUCR $\rightarrow$ local LU-CR factorization; CR $\rightarrow$ local CR factorization; CCR $\rightarrow$ local centered CR factorization; and SCR $\rightarrow$ local stable CR factorization. Table 5..1 gives operation counts for the factorization of the BABD matrix and solution of the BABD system. We have assumed a parallel solution using a number of processors decreasing from $p/2$ (in the first step of reduction) to 1 (in the last). Supposing the factorization and system solution are separate tasks, on a sequential computer $(2n^2+n)(N+1)$ elements of memory are needed for the BABD matrix. Table 5..2 summarizes the memory requirements. We have assumed n is large enough so that lower order powers in n than $\mathcal{O}(n^3)$ may be neglected in the operation counts. Some of these operation counts are also given in [84,101]. For all of the parallel algorithms, the number of transmissions is $\log_2 p\left[t(2n^2)+t(n)\right]$, where $t(k)$ is the time for one transmission of k elements. In fact, Table 5..2 underestimates the memory requirements for a computationally efficient implementation. Keeping rows of successive blocks stored in consecutive locations in cyclic reduction to avoid communication costs necessitates using $3n^2(N/p+\log_2 p)$ additional memory locations. Similarly, the LU and CR based algorithms require $2n^2\log_2 p$ and $n^2\log_2 p$ additional locations, respectively.

Based on flop counts and memory requirements, it is clear that algorithms SCR, CCR, and CR are to be preferred other considerations being equal. However, CR is often unstable while SCR is guaranteed stable. Algorithm CCR may be stable but is without guarantees so should be used with care.

In numerical tests, predictably the CR algorithm has the shortest execution times, but in most cases the computed solution is incorrect. All other algorithms return approximately the same relative error. (In the CCR algorithm, one must find an appropriate size for $S_{j,2}, T_{j,2}$ to avoid instability.) CCR slightly outperforms SCR in efficiency because pivoting is applied to matrices of smaller size. (Numerical examples and comparisons are given in [5,6].) Because of the high cost of permutations (operations not included in Table 5..1),

Table 5..1. Operation counts – nonseparated BCs

	Factorization	Solution
LU	$\frac{23}{3}n^3(N/p+\log_2 p)$	$8n^2(N/p+\log_2 p)$
QR	$\frac{46}{3}n^3(N/p+\log_2 p)$	$15n^2(N/p+\log_2 p)$
LUCR	$\frac{23}{3}n^3(N/p+\log_2 p)$	$8n^2(N/p+\log_2 p)$
CR	$\frac{14}{3}n^3(N/p+\log_2 p)$	$6n^2(N/p+\log_2 p)$
CCR	$\frac{14}{3}n^3(N/p+\log_2 p)$	$6n^2(N/p+\log_2 p)$
SCR	$\frac{14}{3}n^3(N/p+\log_2 p)$	$6n^2(N/p+\log_2 p)$

Table 5..2. Memory requirements – nonseparated BCs

LU	$4n^2(N/p+\log_2 p)+n(N/p+1)$
QR	$4n^2(N/p+\log_2 p)+n(N/p+1)$
LUCR	$4(n^2+n)(N/p+\log_2 p)+n(N/p+1)$
CR	$3n^2(N/p+\log_2 p)+n(N/p+1)$
CCR	$3n^2(N/p+\log_2 p)+n(N/p+1)$
SCR	$3n^2(N/p+\log_2 p)+n(N/p+1)$

the CR algorithms do not have the predicted computational cost advantage over the LU based solvers. So, the "best" algorithm cannot clearly be identified simply by considering flop counts and memory requirements.

5.2. Separated BCs

As in (2..2), the BCs are

$$D_a\mathbf{y}(a)=\mathbf{c}_a,\quad D_b\mathbf{y}(b)=\mathbf{c}_b,\qquad \mathbf{c}_a\in\mathcal{R}^q,\quad \mathbf{c}_b\in\mathcal{R}^{n-q}. \tag{5..11}$$

Using the mesh π and a basic discretization of (2..2), we obtain an ABD linear system, cf. (2..5):

$$\mathcal{A}\mathbf{x}\equiv\begin{pmatrix} D_a & & & & & \\ S_1 & T_1 & & & & \\ & S_2 & T_2 & & & \\ & & \ddots & \ddots & & \\ & & & S_N & T_N & \\ & & & & D_b & \end{pmatrix}\begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{x}_N \end{pmatrix}=\begin{pmatrix} \mathbf{c}_a \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_N \\ \mathbf{c}_b \end{pmatrix}\equiv\mathbf{f}. \tag{5..12}$$

For simplicity, assume $N = kp - 2$. The system can be divided into p parts, assigning the subsystem

$$\begin{pmatrix} D_a & & & \\ S_1 & T_1 & & \\ & \ddots & \ddots & \\ & & S_{k-1} & T_{k-1} \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{k-1} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_a \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{k-1} \end{pmatrix}, \tag{5..13}$$

to the first processor, the subsystem

$$\begin{pmatrix} S_{(i-1)k} & T_{(i-1)k} & & \\ & \ddots & \ddots & \\ & & S_{ik-1} & T_{ik-1} \end{pmatrix} \begin{pmatrix} \mathbf{x}_{(i-1)k-1} \\ \vdots \\ \mathbf{x}_{ik-1} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_{(i-1)k} \\ \vdots \\ \mathbf{b}_{ik-1} \end{pmatrix}, \tag{5..14}$$

to the i^{th} processor, $i = 2, 3, \ldots, p-1$, and the subsystem

$$\begin{pmatrix} S_{(p-1)k} & T_{(p-1)k} & & \\ & \ddots & \ddots & \\ & & S_N & T_N \\ & & & D_b \end{pmatrix} \begin{pmatrix} \mathbf{x}_{(p-1)k-1} \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} \mathbf{b}_{(p-1)k} \\ \vdots \\ \mathbf{b}_N \\ \mathbf{c}_b \end{pmatrix}. \tag{5..15}$$

to the p^{th} processor. For each parallel algorithm, local factorizations of these subsystems produce a reduced system with structure (5..12). The difference from the nonseparated case is that the unknowns $\mathbf{x}_{k-1}, \mathbf{x}_{2k-1}, \ldots, \mathbf{x}_{(p-1)k-1}$, are involved in the reduced system of size $(p-1)n$.

5.2.1. Local CR Factorization on the Normal Equations In [8], the normal equations $\mathcal{A}^T\mathcal{A}\mathbf{y} = \mathcal{A}^T\mathbf{f}$ for (5..12) are formed then cyclic reduction is applied to solve the resulting block tridiagonal system with coefficient matrix

$$\mathcal{A}^T\mathcal{A} = \begin{pmatrix} D_a^T D_a + S_1^T S_1 & S_1^T T_1 & & \\ T_1^T S_1 & T_1^T T_1 + S_2^T S_2 & S_2^T T_2 & \\ & & \ddots & \\ & T_{N-1}^T S_{N-1} & T_{m-1}^T T_{N-1} + S_M^T S_N & S_N^T T_N \\ & & T_N^T S_N & D_b^T D_b \end{pmatrix}. \tag{5..16}$$

The condition number of $\mathcal{A}^T\mathcal{A}$ is the square of that of the ABD matrix $\mathcal{A}$, which can be a major drawback for stability.

Though not explicitly stated in [8], if k is a power of 2, cyclic reduction as in [2,7] does not require communication until the solution phase for the block tridiagonal reduced system of size $(p-1)n \times (p-1)n$. Thus, the number of communications is reduced to $\log_2 p$ (and hence is independent of k or n).

5.2.2. Local LU Factorization In [4,126], LU factorization is used on each block of (5..12). Consider the partition

$$S_j = \begin{pmatrix} S_{j,1} \\ S_{j,2} \end{pmatrix}, \qquad T_j = \begin{pmatrix} T_{j,1} \\ T_{j,2} \end{pmatrix},$$

where $S_{j,1}$ and $T_{j,1}$ have $n-q$ rows (the number of right BCs). On the i^{th} processor, $i = 2, 3, \ldots, p-1$, form the partition

$$\left(\begin{array}{c|ccc|c} S_{(i-1)k,1} & T_{(i-1)k,1} & & & \\ \hline S_{(i-1)k,2} & T_{(i-1)k,2} & & & \\ & S_{(i-1)k+1} & T_{(i-1)k+1} & & \\ & & \ddots & & \\ & & S_{ik-2} & T_{ik-2} & \\ & & & S_{ik-1,1} & T_{ik-1,1} \\ \hline & & & S_{ik-1,2} & T_{ik-1,2} \end{array}\right), \tag{5..17}$$

and sequentially factorize the internal ABD matrix

$$M_i = \left(\begin{array}{ccc} T_{(i-1)k,2} & & \\ S_{(i-1)k+1} & T_{(i-1)k+1} & \\ & \ddots & \\ & S_{ik-2} & T_{ik-2} \\ & & S_{ik-1,1} \end{array}\right). \tag{5..18}$$

On the first processor, substitute D_a for $T_{0,2}$ in M_1, and, on the p^{th} processor, substitute D_b for $S_{N-1,1}$ in M_p, then the decomposition is as in (5..17) without the first row/column for M_1 and the last row/column for M_p. Even if (5..17) is nonsingular, a block in (5..18) may be singular; this problem may be avoided by using row permutations inside the blocks $T_{(i-1)k}$ and S_{ik-1}.

In [126], (5..17) is factored

$$\left(\begin{array}{c|cc} I_{n-q} & T_{(i-1)k,1} & \\ \hline & M_i & \\ \hline & S_{ik-1,2} & I_q \end{array}\right) \left(\begin{array}{c|c|c} \hat{S}_{(i-1)k,1} & & \hat{T}_{(i-1)k,1} \\ \hline V_{(i-1)k,2} & & W_{(i-1)k,2} \\ \vdots & I_{(k-1)n} & \vdots \\ V_{ik-1,1} & & W_{ik-1,1} \\ \hline \hat{S}_{ik-1,2} & & \hat{T}_{ik-1,2} \end{array}\right). \tag{5..19}$$

In [4], it is observed that the factorization

$$\left(\begin{array}{c|ccc|c} \hat{S}_{(i-1)k,1} & V_{(i-1)k,1} & \cdots & V_{ik-1,1} & \hat{T}_{ik-1,1} \\ \hline & I_{(k-1)n} & & & \\ \hline \hat{S}_{(i-1)k,2} & W_{(i-1)k,2} & \cdots & W_{ik-1,2} & \hat{T}_{ik-1,2} \end{array}\right) \left(\begin{array}{c|c|c} I_{n-q} & & \\ \hline S_{(i-1)k,2} & M_i & \\ & & T_{ik-1,1} \\ \hline & & I_q \end{array}\right) \tag{5..20}$$

produces fill-in of half the size for the rows containing $T_{(i-1)k,1}$ and $S_{ik-1,2}$.

5.2.3. Computational Considerations - Parallel ABD Algorithms Tables 5..3 and 5..4 give arithmetic costs and memory requirements, respectively, for the ABD linear system arising from a BVODE (2..2). We assume that there are $N = kp-2$ internal blocks of size $n \times n$. We use the acronyms: CR $\rightarrow$ local CR factorization on the normal equations; $\text{LU}_{PG} \rightarrow$ local LU factorization (5..19); and $\text{LU}_{AP} \rightarrow$ local LU factorization (5..20).

Table 5..3. Operation counts – separated BCs – q left hand BCs

	Factorization	Solution
CR	$\frac{37}{3}n^3(N+2)/p + \frac{25}{3}n^3\log_2 p$	$n^2(14(N+2)/p + 10\log_2 p)$
LU_{PG}	$(\frac{29}{3}n^3 + \frac{1}{2}qn^2 - \frac{1}{2}q^2n)(N+2)/p + \frac{26}{3}n^3\log_2 p$	$8n^2((N+2)/p + \log_2 p)$
LU_{AP}	$(\frac{17}{3}n^3 - 2qn^2 + q^2n)(N+2)/p + \frac{14}{3}n^3\log_2 p$	$6n^2((N+2)/p + \log_2 p)$

Table 5..4. Memory requirements – separated BCs

CR	$n^2(5(N+2)/p + 3\log_2 p) + n(N+2)/p$
LU_{PG}	$4n^2((N+2)/p + \log_2 p) + 2n^2 + n(N+2)/p$
LU_{AP}	$3n^2((N+2)/p + \log_2 p) - n^2 + n(N+2)/p$

6. Iterative Methods for BABD Systems

Direct methods are clearly the algorithms of choice for ABD systems, where it is easy to devise approaches which preserve structure and which are stable. For BABD systems, the choices are less clear. As shown in Section 4.2.1., Gaussian elimination with row interchanges can be unstable and leads to some fill-in (about 50% of the memory needed for the BABD). Alternative sequential approaches discussed in Section 4.2.2. build in the fill-in to achieve stability or proceed by a recursive approach whose stability has not been fully analyzed. Another alternative is to use one of the algorithms for parallel solution outlined in Section 5.. These can be implemented sequentially but all involve fill-in, and some have inferior stability properties. So, given that there are problems of potential instability and of fill-in with direct methods for BABD systems, iterative methods such as preconditioned conjugate gradients provide an alternative approach, especially since they provide an opportunity for parallel computation.

6.1. Preconditioned Conjugate Gradients

For large systems (5..1), we consider conjugate gradients (CG) using $\mathcal{A}$ and $\mathcal{A}^T$ in matrix-vector products. For nonsymmetric matrices, we can apply CG (implicitly) to $\mathcal{A}^T\mathcal{A}\mathbf{x} = \mathcal{A}^T\mathbf{b}$ or $\mathcal{A}\mathcal{A}^T\mathbf{w} = \mathbf{b}$ (with $\mathbf{x} = \mathcal{A}^T\mathbf{w}$).

In [98,99], CG is applied (implicitly) to the normal equations $\mathcal{A}^T\mathcal{A}\mathbf{x} = \mathcal{A}^T\mathbf{b}$ using Algorithm 6.1, a modified version of Algorithm 10.3.1 in [78]. In practice, the number of iterations is roughly proportional to $\sqrt{\mathrm{cond}(\mathcal{A}^T\mathcal{A})}$; see [57]. Numerical tests demonstrate convergence in the predicted maximum of $\mathcal{O}(nN)$ iterations when solving the normal equations, [99].

To accelerate the convergence of CG, a preconditioner $\mathcal{M}$ is needed to gather the eigenvalues of $\mathcal{M}\mathcal{A}^T\mathcal{A}$ near unity. The difficulty lies in finding a preconditioner which permits an efficient parallel implementation. Both the computation and the application of the preconditioner are crucial to the cost of this algorithm and both are generally difficult to

Algorithm 6.1

Preconditioner $\mathcal{M} \simeq (\mathcal{A}^T\mathcal{A})^{-1}$, $j = 0, \mathbf{y}_0 = \mathbf{0}, \mathbf{r}_0 = \mathcal{A}^T\mathbf{b}$
while $||\mathbf{r}_j||/||\mathcal{A}^T\mathbf{f}|| >$ *error tolerance*
$\mathbf{z}_j = \mathcal{M}\mathbf{r}_j$; $j = j + 1$
if $j = 1$ then $\mathbf{p}_1 = \mathbf{z}_0$
else $\beta_j = \mathbf{r}_{j-1}^T\mathbf{z}_{j-1}/\mathbf{r}_{j-2}^T\mathbf{z}_{j-2}$; $\mathbf{p}_j = \mathbf{z}_{j-1} + \beta_j\mathbf{p}_{j-1}$
$\alpha_j = \mathbf{r}_{j-1}^T\mathbf{z}_{j-1}/(\mathcal{A}\mathbf{p}_j)^T(\mathcal{A}\mathbf{p}_j)$
$\mathbf{x}_j = \mathbf{x}_{j-1} + \alpha_j\mathbf{p}_j$; $\mathbf{r}_j = \mathbf{r}_{j-1} - \alpha_j\mathcal{A}^T\mathcal{A}\mathbf{p}_j$
endwhile

parallelize at the block submatrix level.

One possibility uses a factorization of each diagonal block of $\mathcal{A}^T\mathcal{A}$, or $\mathcal{A}\mathcal{A}^T$, as the (parallel) preconditioner. This converges in $\mathcal{O}(nN)$ iterations, essentially the same as CG, [97]. Other widely used, general purpose preconditioners are also unsuccessful in improving convergence.

6.1.1. Approximate Inverse Preconditioner An alternate approach is to investigate the structure of the (dense) inverse of the matrix $\mathcal{A}$ in (5..1). Each $n \times n$ block of $\mathcal{A}^{-1}$ depends on every block of $\mathcal{A}$. In $\mathcal{A}^{-1}$, the blocks with largest elements are generally not within the block structure of $\mathcal{A}$, [125]. The distribution of large elements depends on the ODE and the associated BCs, [99]. For the trapezoidal rule, say, $\mathcal{A}$ consists of blocks originating from the discretization of the BVODE. As $N \to \infty$, $\frac{h_i}{2}A(x_{i-1})$, $\frac{h_i}{2}A(x_i) \to 0$ and we approximate $\mathcal{A}$ by

$$\mathcal{Z} = \begin{pmatrix} -I & I & & & \\ & -I & I & & \\ & & \ddots & \ddots & \\ & & & -I & I \\ B_a & & & & B_b \end{pmatrix}.$$

If $(B_a + B_b)^{-1}$ exists, then

$$\mathcal{Z}^{-1} = \mathrm{diag}((B_a + B_b)^{-1}) \begin{pmatrix} -B_b & -B_b & \cdots & -B_b & I \\ B_a & -B_b & \cdots & -B_b & I \\ B_a & B_a & \cdots & -B_b & I \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ B_a & B_a & \cdots & B_a & I \end{pmatrix}.$$

The approximate inverse preconditioned conjugate gradient algorithm uses the approximate inverse preconditioner $\mathcal{M} = \mathcal{Z}^{-1}(\mathcal{Z}^{-1})^T$ in Algorithm 6.1. It gives remarkably rapid convergence for many examples. However, it has two disadvantages:

(i) $(B_a + B_b)^{-1}$ must exist, which is untrue in many commonly occurring cases;
(ii) no account is taken of the effect of the step sizes h_i nor of the behavior of $A(x)$ with x; hence this preconditioner can be ineffective for realistic mesh distributions.

Using the constant approximate inverse forces the use of $(B_a + B_b)^{-1}$.

In [99], approximate inverse PCG is compared with CG and with the structured LU factorization of [149] on both a Sun 4/490 and a 20 processor Sequent Symmetry (shared

memory) computer. For the chosen test problems, CG converges as expected. Approximate inverse PCG converges in a very small number of iterations. It is competitive with structured LU factorization and can outperform it on large problems when using a large number of processors. The matrix $\mathcal{Z}$ requires only two distinct $n \times n$ blocks of storage and application of $\mathcal{Z}^T\mathcal{Z}$ is easily parallelized. Parallelizing the underlying matrix-vector multiplies in PCG gives linear speedup when the matrices are assembled in parallel using the same block structure [98]. This carries over to approximate inverse PCG, which also achieves close to linear speedup. In [97], the use of better approximations to S_i and R_i in $\mathcal{Z}$ (and hence the preconditioner $\mathcal{M}$) is discussed. Though these approximations sometimes lead to faster convergence, none are uniformly better.

6.1.2. Block Splitting Preconditioner In [43], the condition number of the iteration matrix is reduced via a splitting $\mathcal{A}^T\mathcal{A} = \mathcal{B}^T\mathcal{B} + \mathcal{C}$, where $\mathcal{B}^T\mathcal{B}$ is symmetric positive definite and $\mathcal{C}$ is symmetric. O'Leary [123] suggested splitting $\mathcal{A}^T\mathcal{A}$ by placing the BCs in $\mathcal{C}$ then replacing the right BC with the identity to ensure invertibility, giving the splitting PCG algorithm. Let

$$\mathcal{B} = \begin{pmatrix} S_0 & T_0 & & & \\ & S_1 & T_1 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & T_{N-1} \\ & & & & I \end{pmatrix}. \tag{6..1}$$

Thus, $\mathcal{A}^T\mathcal{A} = \mathcal{B}^T\mathcal{B} + \mathcal{C}$, with

$$\mathcal{C} = \begin{pmatrix} B_a^T B_a & O & \cdots & O & B_a^T B_b \\ O & O & \cdots & O & O \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ O & O & \cdots & O & O \\ B_b^T B_a & O & \cdots & O & B_b^T B_b - I \end{pmatrix}, \tag{6..2}$$

and rank$(\mathcal{C}) = 2n$. The resulting preconditioned system in Algorithm 6.1, $\mathcal{B}^T\mathcal{B}\mathbf{z} = \mathbf{r}$, is solved in two stages, $\mathcal{B}^T\mathbf{w} = \mathbf{r},\ \mathcal{B}\mathbf{z} = \mathbf{w}$, in an obvious block recursive way.

If $\mathcal{A} = \mathcal{I} + \mathcal{U}$ is symmetric positive definite and rank$(\mathcal{U}) \leq s$, at most $nN - s$ eigenvalues of $\mathcal{A}$ are unity. Hence $(\mathcal{B}^T\mathcal{B})^{-1}(\mathcal{A}^T\mathcal{A})$ has at least $nN - s$ unit eigenvalues [100]. If $\mathcal{M}(\mathcal{A}^T\mathcal{A})$ has $s < n(N+1)$ distinct eigenvalues, in exact arithmetic splitting PCG converges in s iterations [43]. Since $(\mathcal{B}^T\mathcal{B})^{-1}(\mathcal{A}^T\mathcal{A})$ has at most $2n+1$ distinct eigenvalues, splitting PCG using $\mathcal{M} = (\mathcal{B}^T\mathcal{B})^{-1}$ converges in at most $2n+1$ iterations. Neither $\mathcal{A}^T\mathcal{A}$ nor $\mathcal{B}^T\mathcal{B}$ should be formed explicitly; only matrix-vector products with $\mathcal{A}$ and $\mathcal{A}^T$, or $\mathcal{B}$ and $\mathcal{B}^T$, are needed. For computational results for splitting PCG, see [100]. It works as predicted on many discretized BVODEs but, for some, $\mathcal{M}$ is ill-conditioned and it does not converge. Exponential growth factors are observed which compare in size and origin with those obtained in discretizations of example (4..3).

One of us (GLK) has experimented with *qmrpack* [73,74] to check the results of splitting PCG. Since *qmrpack* does not permit an internal preconditioner, $\mathcal{B}^{-1}$ is computed and applied externally. There are differences in convergence behavior between splitting PCG and the three-step look-ahead CG algorithm implemented in *qmrpack*, particularly:

(i) where splitting PCG converges quickly, so does *qmrpack* using a similar number of matrix-vector multiplies;

(ii) where splitting PCG diverges, so does *qmrpack*;
(iii) where splitting PCG converges slowly, *qmrpack* terminates (seemingly successfully) after a small number of iterations. The "converged" result has a large residual and the solution is inaccurate, though the scaled residual used internally by *qmrpack* is small.

7. Conclusions

We have outlined a variety of discretizations of ordinary and partial differential equations which lead to almost block diagonal and bordered almost block diagonal linear systems, whose precise form depends on the discretization and the boundary conditions. The solution techniques that we described are designed mainly for the generic problem. Where a major gain in efficiency is possible, as in a multiple shooting discretization for boundary value ordinary differential equations and in certain partial differential equation applications, we have discussed how to exploit it. The references include several that describe software development for almost block diagonal and bordered almost block diagonal linear systems, and a significant proportion of these have associated publicly available software for which we have given a source. It seems that sequential algorithms for the generic almost block diagonal problem need little further study or software development, but all the other areas discussed here are still open. As new applications arise, there will be a need to study the merits of further refinements of the algorithms for these particular cases. Also, as computers, software components, such as the BLAS, and parallel and object oriented languages, develop, there will be a need to revisit the analysis and algorithms.

Acknowledgements

The authors acknowledge the invaluable assistance of Karin Remington who provided the figures. We also thank our colleagues Bernard Bialecki, Patrick Keast, Gerald Moore and Paul Muir for their help in providing information used in this paper. The comments of an anonymous referee were particularly helpful in a revision of the paper

REFERENCES

1. J.S. Albuquerque and L.T. Biegler. Decomposition algorithms for on-line estimation with nonlinear DAE models. Report EDRC 06-192-95, Carnegie Mellon University Engineering Design Research Center, 1995.
2. P. Amodio, L. Brugnano and T. Politi. Parallel factorizations for tridiagonal matrices. *SIAM J. Numer. Anal.*, **30**, 1993, 813-823.
3. P. Amodio and N. Mastronardi. A parallel version of the cyclic reduction algorithm on a hypercube. *Parallel Comput.*, **19**, 1993, 1273-1281.
4. P. Amodio and M. Paprzycki. Parallel solution of almost block diagonal systems on a hypercube. *Lin. Alg. Applic.*, **241-243**, 1996, 85-103.
5. P. Amodio and M. Paprzycki. On the parallel solution of almost block diagonal systems. *Control & Cybernetics*, **25**, 1996, 645-656.
6. P. Amodio and M. Paprzycki. Recent advances in the parallel solution to almost block diagonal

systems. In Vol. 2 Proc. ICIAM '95 *Applied Analysis* (eds. O. Mahrenholtz and R. Mennicken). *ZAMM*, **76**, 1996, 1-4.
7. P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODE's. *SIAM J. Sci. Comput.*, **18**, 1997, 56-68.
8. U.M. Ascher and S.Y.P. Chan. On parallel methods for boundary value ODEs. *Computing*, **46**, 1991, 1-17.
9. 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**, (eds. B. Childs et al.). Springer-Verlag, New York, 1979, 164-185. Code *colsys* is available from library *ode* on *netlib*.
10. U.M. Ascher, J. Christiansen and R.D. Russell. Collocation software for boundary value ODE's. *ACM Trans. Math. Soft.* **7**, 1981, 209-229.
11. U.M. Ascher, R.M.M. Mattheij and R.D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* (2nd ed.). SIAM, Philadelphia, 1995.
12. U.M. Ascher, S. Pruess and R.D. Russell. On spline basis selection for solving differential equations. *SIAM J. Numer. Anal.*, **20**, 1983, 121-142.
13. U.M. Ascher and R.D. Russell. Reformulation of boundary value problems into 'standard form'. *SIAM Rev.*, **23**, 1981, 238-254.
14. G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value solver. *SIAM J. Sci. Stat. Comp.*, **9**, 1987, 483-500. Code *colnew* is available from library *ode* on *netlib*.
15. K.R. Bennett. *Parallel collocation methods for boundary value problems*. Ph.D. thesis, Department of Mathematics, University of Kentucky, 1991. Code *pabbpack*, a parallel implementation of the SLU algorithm of S.J. Wright [147], is available by anonymous ftp from directory /pub/karin/pabbpack/ at math.nist.gov. (This version was prepared for parallel processors that are no longer widely available; it uses machine-specific parallel directives.)
16. K.R. Bennett and G. Fairweather. A parallel boundary value ODE code for shared-memory machines. *Int. J. High Speed Comp.*, **4**, 1992, 71-86. 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. F.W. Beaufait and G.W. Reddien. Midpoint difference method for analyzing beam structures. *Comput. Struct.*, **8**, 1978, 745-751.
18. D. Bhattacharyya, M. Jevtitch, J.T. Schrodt and G. Fairweather. Prediction of membrane separation characteristics by pore distribution measurements and surface force-pore flow model. *Chem. Eng. Commun.*, **42**, 1986, 111-128.
19. B. Bialecki. An alternating direction implicit method for orthogonal spline collocation linear systems. *Numer. Math.*, **59**, 1991, 413-429.
20. B. Bialecki. Cyclic reduction and FACR methods for piecewise Hermite bicubic orthogonal spline collocation. *Numer. Alg.*, **8**, 1994, 167-184.
21. B. Bialecki. A fast solver for the orthogonal spline collocation solution of the biharmonic Dirichlet problem on rectangles. Submitted.
22. B. Bialecki and G. Fairweather. Matrix decomposition algorithms for separable elliptic boundary value problems in two space dimensions. *J. Comp. Appl. Math.*, **46**, 1993, 369-386.
23. B. Bialecki and G. Fairweather. Matrix decomposition algorithms for orthogonal spline collocation for separable elliptic boundary value problems. *SIAM J. Sci. Comp.*, **16**, 1995, 330-347.
24. B. Bialecki, G. Fairweather and K.R. Bennett. Fast direct solvers for piecewise Hermite bicubic orthogonal spline collocation equations. *SIAM J. Numer. Anal.*, **29**, 1992, 156-173.
25. B. Bialecki and R.I. Fernandes. Orthogonal spline collocation Laplace-modified and alternating-direction methods for parabolic problems on rectangles. *Math. Comp.*, **60**, 1993, 545-573.
26. B. Bialecki and R.I. Fernandes. An orthogonal spline collocation alternating direction implicit Crank-Nicolson method for linear parabolic problems on rectangles. *SIAM J. Numer. Anal.*, **36**, 1999, 1414-1434
27. B. Bialecki and R.I. Fernandes. An orthogonal spline collocation alternating direction implicit method for linear second order hyperbolic problems on rectangles. Submitted.
28. B. Bialecki and K.A. Remington. Fourier matrix decomposition methods for the least squares solution of singular Neumann and periodic Hermite bicubic collocation problems. *SIAM J. Sci.*

Comp., **16**, 1995, 431-451.
29. H-G. Bock. Recent advances in parameter identification techniques for O.D.E. In *Numerical Treatment of Inverse Problems in Differential Equations*, *Progress in Scientific Computing*. **2** (eds. P. Deuflhard and E. Hairer) Birkhäuser, Boston, 1983, 95-121.
30. Z. Bohte. Bounds for rounding errors in Gaussian elimination. *J. Inst. Math. Appl.*, **16**, 1975, 790-805.
31. C. de Boor. A package for calculating with B-splines. *SIAM J. Numer. Anal.*, **14**, 1977, 441-472.
32. C. de Boor. *A Practical Guide to Splines*, Springer-Verlag, New York, 1978. Software available as the *spline toolbox* from Mathworks.
33. C. de Boor and R. Weiss. SOLVEBLOK: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Softw.*, **6**, 1980, 80-87.
34. C. de Boor and R. Weiss. Algorithm 546: SOLVEBLOK. *ACM Trans. Math. Softw.*, **6**, 1980, 88-91.
35. R.W. Brankin and I. Gladwell. Codes for almost block diagonal systems. *Comput. Math Applic.*, **19**, 1990, 1-6. Code *f01lhf* is available in the *NAG* Fortran 77 library.
36. J.R. Cash. The numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections; Part 1: a survey and comparison of some one step formulae. *Comput. Math. Applic.*, **12**, 1986, 1029-1048.
37. J.R. Cash. The numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections; Part 2: the development and analysis of highly stable deferred correction formulae. *SIAM J. Numer. Anal.*, **25**, 1988, 862-882.
38. J.R. Cash, G. Moore and R.W. Wright. An automatic continuation strategy for the solution of singularly perturbed linear boundary value problems. *J. Comp. Phys.*, **122**, 1995, 266-279. Code *colmod* available from library *ode* on *netlib*.
39. J.R. Cash, G. Moore and R.W. Wright. An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems, submitted. Code *acdc* available from library *ode* on *netlib*.
40. J.R. Cash and A. Singhal. High order methods for the numerical solution of two point boundary value problems. *BIT*, **22**, 1982, 184-199.
41. J.R. Cash and M.H. Wright. Implementation issues in solving nonlinear equations for two-point boundary value problems. *Computing*, **45**, 1990, 17-37.
42. J.R. Cash and M.H. Wright. A deferred correction method for nonlinear two-point boundary value problems. *SIAM J. Sci. Stat. Comp.*, **12**, 1991, 971-989. Code *twpbvp* available from library *ode* on *netlib*.
43. P. Concus, G.H. Golub, and D.P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In *Sparse Matrix Computations* (eds. J.R. Bunch and D.J. Rose). Academic Press, New York, 1976, 309-332.
44. K.D. Cooper. Domain-imbedding alternating direction method for linear elliptic equations on irregular regions using collocation. *Numer. Meth. Partial Diff. Equ.*, **9**, 1993, 93-106.
45. K.D. Cooper, K.M. McArthur and P.M. Prenter. Alternating direction collocation for irregular regions. *Numer. Meth. Partial Diff. Equ.*, **12**, 1996, 147-159.
46. K.D. Cooper and P.M. Prenter. A coupled double splitting ADI scheme for the first biharmonic using collocation. *Numer. Meth. Partial Diff. Equ.*, **6**, 1990, 321-333.
47. K.D. Cooper and P.M. Prenter. Alternating direction collocation for separable elliptic partial differential equations. *SIAM J. Numer. Anal.*, **28**, 1991, 711-727.
48. 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, 1992. Code *l3abdsol* is available from library *linalg* on *netlib*.
49. C. Cyphers, M. Paprzycki and A. Karageorghis. High performance solution of partial differential equations discretized using a Chebyshev spectral collocation method. *J. Comp. Appl. Math.*, **69**, 1996, 71-80.
50. 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 Trans. Math. Softw.*, **9**, 1983, 358-375.
51. 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 Trans. Math. Softw.*, **9**, 1983, 376-380.
52. E. Doedel, H.B. Keller and J. Kernevez. Numerical analysis and control in bifurcation problems, (I) Bifurcations in finite dimensions. *Int. J. Bif. Chaos*, **1**, 1991, 493-520.
53. E. Doedel, H.B. Keller and J. Kernevez. Numerical analysis and control in bifurcation problems, (II) Bifurcations in infinite dimensions. *Int. J. Bif. Chaos*, **2**, 1992, 745-772.
54. E. Doedel, X.J. Wang and T.F. Fairgrieve. AUTO94: software for continuation and bifurcation problems in ordinary differential equations. Technical Report CRPC-95-1, Center for Research on Parallel Computing, California Institute of Technology, 1995. Code *auto94* and a more recent version *auto97* are available by contacting the author at *doedel@cs.concordia.ca*.
55. 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 *netlib*, in the *NAG* libraries, and, in optimized form, from many computer manufacturers.
56. J. Dongarra, J. Du Croz, S. Hammarling and R.J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, **14**, 1988, 1-17. The level 2 *BLAS* are available on *netlib*, in the *NAG* libraries, and, in optimized form, from many computer manufacturers.
57. J. Dongarra, I.S. Duff, D.C. Sorensen and H.A. Van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, 1991.
58. J. Douglas Jr. and T. Dupont. *Collocation Methods for Parabolic Equations in a Single Space Variable*. Lecture Notes in Mathematics **385**. Springer-Verlag, New York, 1974.
59. C. Dun, M. Hegland and M.R. Osborne. Stable, parallel solution methods for tridiagonal systems of linear equations. In *Computational Techniques & Applications: CTAC95*, (eds. R.L. May and A.K. Easton). World Scientific, Singapore, 1996, 267-274.
60. W.R. Dyksen. Tensor product generalized ADI methods for separable elliptic problems. *SIAM J. Numer. Anal.*, **24**, 1987, 59-76.
61. W.H. Enright and P.H. Muir. Efficient classes of Runge-Kutta methods for two-point boundary value problems. *Computing*, **37**, 1986, 315-344.
62. W.H. Enright and P.H. Muir. Runge-Kutta software with defect control for boundary value ODES. *SIAM J. Sci. Comp.*, **17**, 1996, 479-497.
63. G. Fairweather. *Finite Element Galerkin Methods for Differential Equations*. Lecture in Notes Pure and Applied Mathematics **34**. Marcel Dekker, New York, 1978.
64. G. Fairweather. A note on the efficient implementation of certain Padé methods for linear parabolic problems. *BIT*, **18**, 1978, 106-109.
65. G. Fairweather, K.R. Bennett and B. Bialecki. Parallel matrix decomposition algorithms for separable elliptic boundary value problems. In *Computational Techniques and Applications: CTAC-91*. Proc. 1991 International Conference on Computational Techniques and Applications, Adelaide, South Australia, July 1991, (eds. B.J. Noye et al). Computational Mathematics Group, Australian Mathematical Society, Adelaide, Australia, 1992, 63-74.
66. 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. Code *rowcol.f* is available by anonymous ftp from directory /keast/abd_solvers at ftp.cs.dal.ca.
67. 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**, 1984, 314-326.
68. G. Fairweather, J.C. López-Marcos and A. Boutayeb. Orthogonal spline collocation for a quasilinear nonlocal parabolic problem. Preprint, 2000.
69. G. Fairweather and R.D. Saylor. The reformulation and numerical solution of certain nonclassical initial-boundary value problems. *SIAM J. Sci. Stat. Comp.*, **12**, 1991, 127-144.
70. R.I. Fernandes. Efficient orthogonal spline collocation methods for solving linear second order hyperbolic problems on rectangles. *Numer. Math.*, **77**, 1997, 223-241.
71. R.I. Fernandes and G. Fairweather. Analysis of alternating direction collocation methods for parabolic and hyperbolic problems in two space variables. *Numer. Meth. Partial Diff. Equ.*, **9**, 1993, 191-211.
72. R. Fourer. Staircase matrices and systems. *SIAM Rev.*, **26**, 1984, 1-70.
73. R.W. Freund, G.H. Golub, and N.M. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, **1**, 1991, 57-100.

74. R.W. Freund, M. Gutknecht and N.M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. Technical Report 91.09, RIACS, NASA Ames Research Center, Moffett Field, 1991. The code *qmrpack* is available in directory *qmr* in library *linalg* of *netlib*.
75. I. Gladwell and R.I. Hay. Vector- and parallelization of ODE BVP codes. *Parallel Comput.*, **12**, 1989, 343-350.
76. I. Gladwell and M. Paprzycki. Parallel solution of almost block diagonal systems on the CRAY Y-MP using level 3 BLAS. *J. Comp. Appl. Math.*, **45**, 1993, 181-189.
77. I. Gladwell and R.M. Thomas. Efficiency of methods for second order problems. *IMA J. Numer. Anal.*, **10**, 1990, 181-207.
78. G.H. Golub and C.F. Van Loan. *Matrix Computations* (3rd ed). The John Hopkins University Press, Baltimore, MD, 1996.
79. W. Govaerts. Stable solvers and block elimination for bordered systems. *SIAM J. Matrix Anal. Applic.*, **12**, 1991, 469-483.
80. W. Govaerts and J.D. Pryce. Mixed block elimination for linear systems with wider borders. *IMA J. Numer. Anal.*, **13**, 1993, 161-180. Code *bemw* is available from library *linalg* on *netlib*.
81. S. Gupta. An adaptive boundary value Runge-Kutta solver for first order boundary value problems. *SIAM J. Numer. Anal.*, **22**, 1985, 114-126.
82. R.I. Hay and I. Gladwell. Solving almost block diagonal linear equations on the CDC Cyber 205. Numerical Analysis Report 98, Department of Mathematics, University of Manchester, 1987.
83. M. Hegland and M.R. Osborne. Wrap-around partitioning for block-bidiagonal linear systems. *IMA J. Numer. Anal.*, **18**, 1998, 373-384.
84. 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, 1991.
85. K.R. Jackson, R. Pancer, and P.H. Muir. Runge-Kutta Software for the Parallel Solution of Boundary Value ODEs. Preprint, 1999.
86. A. Karageorghis. The numerical solution of laminar flow in a re-entrant tube geometry by a Chebyshev spectral element collocation method. *Comput. Methods Appl. Mech. Engng*, **100**, 1992, 339-358.
87. A. Karageorghis and M. Paprzycki. An efficient direct method for fully conforming spectral collocation schemes. *Numer. Alg.*, **12**, 1996, 309-319.
88. A. Karageorghis and T.N. Phillips. On efficient direct methods for conforming spectral domain decomposition techniques. *J. Comp. Appl. Math.*, **33**, 1990, 141-155.
89. P. Keast. Private communication, 1997; code *transcolrow.f* is available at http://www.mscs.dal.ca/~keast/leq/
90. P. Keast. Private communication, 1997; code *complexcolrow.f* is available at http://www.mscs.dal.ca/~keast/leq/
91. P. Keast and G. Fairweather. Private communication, 1997; code *lampak.f* is available http://www.mscs.dal.ca/~keast/leq/
92. 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**, 1983, 499-518.
93. P. Keast and P.H. Muir. Algorithm 688: EPDCOL: a more efficient PDECOL code. *ACM Trans. Math. Softw.*, **17**, 1991, 153-166.
94. H.B. Keller. A new difference scheme for parabolic equations. In *Numerical Solution of Differential Equations - II* (ed. B. Hubbard). Academic Press, New York, 1971, 327-350.
95. H.B. Keller. *Numerical Methods for Two Point Boundary Value Problems*. Dover, New York, 1992.
96. H.B. Keller and A.D. Jepson. Steady state and periodic solution paths: their bifurcations and computations. In *Numerical Methods for Bifurcation Problems* (eds. T. Küpper et al.), *International Series of Numerical Mathematics*. **70**, Birkhäuser, Boston, 1984, 219-246.
97. G.L. Kraut. *Parallel direct and iterative methods for boundary value problems*. Ph.D. thesis, Department of Mathematics, Southern Methodist University, 1993.
98. G.L. Kraut and I. Gladwell. Parallel methods for boundary value problem linear algebra. In *Proc. Sixth SIAM Conf. on Parallel Processing for Scientific Computing*. SIAM, Philadelphia, 1993, 647-651.

99. G.L. Kraut and I. Gladwell. Iterative parallel methods for boundary value problems. In *Proc. Fifth IEEE Symp. on Parallel and Distributed Processing*. SIAM, Philadelphia, 1993, 134-140.
100. G.L. Kraut and I. Gladwell. Convergence and instability in PCG methods for bordered systems. *Comput. Math. Applic.*, **30**, 1995, 101-109.
101. G.L. Kraut and I. Gladwell. Cost of stable algorithms for bordered almost block diagonal systems. In Vol. 1 Proc. ICIAM'95, *Numerical Analysis, Scientific Computation and Computer Science* (eds. G. Alefeld et al.). *ZAMM*, **76**, 1996, 151-154.
102. D.C. Lam. *Implementation of the box scheme and model analysis of diffusion-convection equations*. Ph.D. thesis, University of Waterloo, Waterloo, Canada, 1974.
103. M. Lentini, M.R. Osborne and R.D. Russell. The close relationships between methods for solving two-point boundary value problems. *SIAM J. Numer. Anal.*, **22**, 1985, 280-309.
104. M. Lentini and V. Pereyra. An adaptive finite-difference solver for non-linear two-point boundary problems with mild boundary layers. *SIAM J. Numer. Anal.*, **14**, 1977, 91-111.
105. M. Lentini and V. Pereyra. PASVA3: An adaptive finite difference FORTRAN program for first order nonlinear boundary value problems. In *Codes for Boundary Value Problems in Ordinary Differential Equations, Springer-Verlag Lecture Notes in Computer Science* **76** (eds. B. Childs et al). Springer-Verlag, New York, 1979, 67-88. Code *pasva3* is available as *d02raf* in the *NAG* Fortran 77 library.
106. B. Li, G. Fairweather and B. Bialecki. Discrete-time orthogonal spline collocation schemes for Schrödinger equations in two space variables. *SIAM J. Numer. Anal.*, **35**, 1998, 453-477.
107. B. Li, G. Fairweather and B. Bialecki. Discrete-time orthogonal spline collocation schemes for vibration problems. Submitted.
108. Z. Lou. *Orthogonal spline collocation for biharmonic problems*. Ph.D. thesis, Department of Mathematics, University of Kentucky, 1996.
109. Z. Lou, B. Bialecki and G. Fairweather. Orthogonal spline collocation methods for biharmonic problems. *Numer. Math.*, **80**, 1998, 267-303.
110. N.K. Madsen and R.F. Sincovec. Algorithm 540. PDECOL: general collocation software for partial differential equations. *ACM Trans. Math. Softw.*, **5**, 1979, 326-351.
111. F. Majaess and P. Keast. Algorithms for the solution of linear systems arising from monomial spline basis functions. Dalhousie University, Computer Science Division, Technical Report 1987CS-11, 1987.
112. F. Majaess, P. Keast and G. Fairweather. Packages for solving almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. In *Scientific Software Systems* (eds. J.C. Mason and M.G. Cox). Chapman and Hall, London, 1990, 47-58.
113. F. Majaess, P. Keast and G. Fairweather. The solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. *ACM Trans. Math. Softw.*, **18**, 1992, 193-204.
114. 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**, 1992, 205-210. Revised code *abdpack.f* available by anonymous ftp from directory /keast/abd_solvers at ftp.cs.dal.ca, solves a wider range of problems than the published ABDPACK.
115. S. A. V. Manickam, A. K. Pani and S. K. Chung. A second-order splitting combined with orthogonal cubic spline collocation method for the Rosenau equation. *Numer. Methods Partial Diff. Equ.*, **14**, 1998, 695-716.
116. A. V. Manickam, K. M. Moudgalya and A. K. Pani. Second-order splitting combined with orthogonal cubic spline collocation method for the Kuramoto-Sivashinsky equation. *Comput. Math. Appl.*, **35**, 1998, 5-25.
117. R.M.M. Mattheij and S.J. Wright. Parallel stable compactification for ODE with parameters and multipoint conditions. *Appl. Numer. Math.*, **13**, 1993, 305-333.
118. G. Moore. Computation and parametrization of periodic and connecting orbits. *IMA J. Numer. Anal.*, **15**, 1995, 245-263.
119. G. Moore. Geometric methods for computing invariant manifolds. *Appl. Numer. Math.*, **17**, 1995, 311-318.
120. P.H. Muir. *Implicit Runge-Kutta methods for two-point boundary value problems*. Ph.D. thesis,

University of Toronto, Department of Computer Science, Technical Report 175/84, 1984.
121. P.H. Muir and K. Remington. A parallel implementation of a Runge-Kutta code for systems of nonlinear boundary value ODEs. *Cong. Numer.*, **99**, 1994, 291-305.
122. 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 equations. In *Numerical Analysis, A.R. Mitchell 75th Birthday Volume* (eds. D.F. Griffiths and G.A. Watson). World Scientific, Singapore, 1996, 207-223.
123. D.P. O'Leary. Private communication.
124. R.N. Pancer and K.R. Jackson. The parallel solution of almost block diagonal systems arising in numerical methods for BVPs in ODEs. In *Proc.* 15^{th} *IMACS World Cong.*, Vol. 2. Berlin, 1997, 57-62.
125. M. Paprzycki. *Parallelization of boundary value problem software*. Ph.D. thesis, Department of Mathematics, Southern Methodist University, 1990.
126. M. Paprzycki and I. Gladwell. Solving almost block diagonal systems on parallel computers. *Parallel Comput.*, **17**, 1991, 133-153.
127. M. Paprzycki and I. Gladwell, Solving almost block diagonal systems using level 3 BLAS. In *Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing*. SIAM, Philadelphia, 1992, 52-62.
128. R. Pozo and K. Remington. Fast three-dimensional elliptic solvers on distributed network clusters. In *Parallel Computing: Trends and Applications* (eds. G. Joubert et al.). Elsevier, Amsterdam, 1994, 201-208.
129. J.K. Reid and A. Jennings. On solving almost block diagonal (staircase) linear systems. *ACM Trans. Math. Softw.*, **10**, 1984, 196-201.
130. M. P. Robinson. Orthogonal spline collocation solution of nonlinear Schrödinger equations. In *Mathematics of Computation 1943–1993: a half-century of computational mathematics, Proc. Sympos. Appl. Math.*, **48**, Amer. Math. Soc., Providence RI, 1994, 355–360.
131. M. P. Robinson. The solution of nonlinear Schrödinger equations using orthogonal spline collocation. *Comput. Math. Appl.*, **33**, 1997, 39–57. [Corrigendum: *Comput. Math. Appl.*, **35**, 1998, 151].
132. M.P. Robinson and G. Fairweather. An orthogonal spline collocation method for the numerical solution of underwater acoustic wave propagation problems. In *Computational Acoustics*, Vol. 2, (eds. D. Lee et al.). Elsevier, Amsterdam, 1993, 339-353.
133. M.P. Robinson and G. Fairweather. Orthogonal cubic spline collocation solution of underwater acoustic wave propagation problems. *J. Comp. Acoust.*, **1**, 1993, 355-370.
134. M.P. Robinson and G. Fairweather. Orthogonal spline collocation methods for Schrödinger-type problems in one space variable. *Numer. Math.*, **68**, 1994, 355-376.
135. W. Sun. Orthogonal collocation solution of biharmonic equations. *Intern. J. Computer Math.*, **49**, 1993, 221-232.
136. W. Sun. A high order direct method for solving Poisson's equation in a disc. *Numer. Math.*, **70**, 1995, 501-506.
137. W. Sun. Iterative algorithms for orthogonal spline collocation linear systems. *SIAM J. Sci. Comput.*, **16**, 1995, 720-737.
138. W. Sun. Block iterative algorithms for solving Hermite bicubic collocation equations. *SIAM J. Numer. Anal.*, **33**, 1996, 589-601.
139. W. Sun. Fast algorithms for high-order spline collocation systems. *Numer. Math.*, **81**, 1998, 143-160.
140. W. Sun and N.G. Zamani. A fast algorithm for solving the tensor product collocation equations. *J. Franklin Institute*, **326**, 1989, 295-307.
141. M. van Veldhuizen. A note on partial pivoting and Gaussian elimination. *Numer. Math.*, **29**, 1977, 1-10.
142. J.M. Varah. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal.*, **13**, 1976, 71-75.
143. R. Weiss. The application of implicit Runge-Kutta and collocation methods to boundary value problems. *Math. Comp.*, **28**, 1974, 449-464.
144. K. Wright. Some relationships between implicit Runge-Kutta, collocation and Lanczos τ methods and their stability properties. *BIT*, **20**, 1970, 217-227.
145. K. Wright. Parallel treatment of block-bidiagonal matrices in the solution of ordinary differ-

ential boundary value problems. *J. Comp. Appl. Math.*, **45**, 1993, 191-200.
146. R.W. Wright, J. Cash and G. Moore. Mesh selection for stiff two-point boundary value problems. *Numer. Alg.*, **7**, 1994, 205-224.
147. S.J. Wright. Stable parallel algorithms for two-pointboundary value problems. *SIAM J. Sci. Stat. Comp.*, **13**, 1992, 742-764.
148. S.J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Stat. Comp.*, **14**, 1993, 231-238.
149. S.J. Wright. Stable parallel elimination for boundary value ODE's. *Numer. Math.*, **67**, 1994, 521-536.
150. S.J. Wright and V. Pereyra. Adaptation of a two-point boundary value problem solver to a vector-multiprocessor environment. *SIAM J. Sci. Stat. Comp.*, **11**, 1990, 425-449.
151. P.Y. Yalamov and M. Paprzycki. Stability and performance analysis of a block elimination solver for bordered linear systems. *IMA J. Numer. Anal.*, **19**, 1999, 335-348.