An algorithm for the solution of Bordered ABD linear systems arising from BVPs

Pierluigi Amodio [a], Ian Gladwell [b], Giuseppe Romanazzi [a]

[a] *Dipartimento di Matematica, Università di Bari, I-70125 Bari, Italy*

[b] *Department of Mathematics, SMU, Dallas, TX 75275, USA*

Abstract

We consider the solution of linear systems whose coefficient matrices having a Bordered ABD structure. This kind of system arises in the discretization of BVPs for ordinary and partial differential equations with non-separated boundary conditions. The aim of this paper is to test the Fortran 90 package *BABDCR*, based on a cyclic reduction algorithm, within the BVP code *MIRKDC*. Actually, this code uses *COLROW* which is designed to solve ABD systems, and hence *MIRKDC* only deals with separated boundary conditions. Comparisons between the two implementations are performed. Finally, *BABDCR* is implemented on parallel architectures for an eventual test in *PMIRKDC*.

Key words: Boundary Value Problems, Linear systems solution, Bordered Almost Block Diagonal matrices, Cyclic Reduction

1 Introduction

Almost Block Diagonal (ABD) and Bordered Almost Block Diagonal (BABD) [1] linear systems arise in discretizations of Boundary Value Problems (BVPs) with, respectively, separated and non-separated boundary conditions, for ordinary and partial differential equations. The coefficient matrices associated with ABD linear systems are sparse and characterized by as follows: the nonzero elements are grouped in block rows, there is no intersection between the nonzero columns of two nonconsecutive block rows; and, the main diagonal entries always lie inside the nonzero blocks. BABD matrices also satisfy: the first (or

Email addresses: amodio@dm.uniba.it (Pierluigi Amodio), gladwell@seas.smu.edu (Ian Gladwell), romanazzi@dm.uniba.it (Giuseppe Romanazzi).

the last) block row has an additional block in the right-upper (left-lower) corner whose columns only intersect the nonzero columns of the last (first) block row. In Figs. 1 and 2 we show the ABD and BABD linear system structures most frequently arising in BVP solvers.

$$\begin{pmatrix} B_{top} & & & & \\ S_1 & R_1 & & & \\ & S_2 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & R_N \\ & & & & B_{bot} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ x_{N+1} \end{pmatrix} = \begin{pmatrix} d_a \\ f_1 \\ \vdots \\ f_N \\ d_b \end{pmatrix}$$

Fig. 1. ABD linear system with blocks $S_i, R_i \in \mathbb{R}^{m\times m}$, $B_{top} \in \mathbb{R}^{m_0\times m}$ and $B_{bot} \in \mathbb{R}^{(m-m_0)\times m}$, and vectors $f_i, x_i \in \mathbb{R}^m$, $d_a \in \mathbb{R}^{m_0}$ and $d_b \in \mathbb{R}^{m-m_0}$.

$$\begin{pmatrix} B_a & & & & B_b \\ S_1 & R_1 & & & \\ & S_2 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_N & R_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ x_{N+1} \end{pmatrix} = \begin{pmatrix} d \\ f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

Fig. 2. BABD linear system with blocks $S_i, R_i, B_a, B_b \in \mathbb{R}^{m\times m}$, and vectors $x_i, f_i, d \in \mathbb{R}^m$.

1.1 Solvers for ABD and BABD systems

Because of its relevance in BVP codes, the solution of ABD systems has been the subject of long-term research, see [1,11]. The first ABD code was *SOLVEBLOK* [8]; as for standard LU factorization applied to banded systems, it requires fill-in to ensure stability. The alternate row and column stable elimination, called Varah's procedure [16], exploited the structure of the ABD matrices to avoid fill-in. The packages *COLROW* and *ARCECO* in [9] are based on a modified version of Varah's procedure. *COLROW* solves ABD linear systems with row blocks of the same dimension (Fig. 1) and *ARCECO* solves general ABD linear systems with blocks of varying dimensions. Numerical experiments in [9] demonstrate their effectiveness and their superiority over *SOLVEBLOK*, that is less storage and execution time, on systems arising from BVPs. Successively, several modifications have been proposed to these packages to deal with more specific structures of the coefficient matrix (see [1] and the references therein).

In what follows, bear in mind that BABD systems can be recast as ABD systems of double size (see Section 4). In Section 2 we briefly show the *BABDCR* package [3]. Other packages solving BABD systems, as for example *RSCALE* [13], are designed for systems arising from BVPs and are contained in such codes (see [14]).

1.2 ABD and BABD solvers in BVODE packages

Some nonlinear BVP packages employ ABD packages. The BVP code *COLSYS* [4] uses *SOLVEBLOK* to solve ABD linear systems arising from using OSC at Gauss points with B-spline bases. *COLNEW* [5] uses a modified version of *SOLVEBLOK* to solve ABD linear systems arising from using OSC at Gauss points with monomial spline bases. The Mono Implicit Runge Kutta (MIRK) code with defect control *MIRKDC* [10] uses *COLROW* as a solver for ABD systems. Modified versions of *COLROW* are used in *TWPBVP* [7], a deferred correction code, in *COLMOD* [15], a modified version of *COLNEW*, and in *ACDC* [6], which uses automatic continuation and OSC at Lobatto points to solve singularly perturbed BVODEs. The code *PMIRKDC*, a parallel version of *MIRKDC*, uses the package *RSCALE* which has a good parallel implementation.

In this paper we propose *BABDCR* as an alternative to *COLROW* in the solution of BABD systems and *RSCALE* in a parallel environment. In Section 3 we derive a modified version of *MIRKDC* which is effective to solve BVPs with non-separated boundary conditions. Numerical tests are presented in Section 4. In Section 5 we analyze some numerical results obtained on a distributed memory machine.

2 The *BABDCR* package

Based on an idea in [2], the *BABDCR* package (see [3]) solves BABD systems with the special structure in Fig. 2. The underlying algorithm cyclically reduces the coefficient matrix to derive systems of lower dimension with the same BABD structure. Suppose that the coefficient matrix in Fig. 2 is nonsingular; we reduce each pair of block row equations, for $i = 2j,\ j = 1, 2, \ldots, \lfloor N/2 \rfloor$,

$$\begin{pmatrix} S_{i-1} & R_{i-1} & & \\ & & S_i & R_i \end{pmatrix} \begin{pmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{pmatrix} = \begin{pmatrix} f_{i-1} \\ f_i \end{pmatrix} \tag{1}$$

into one block row equation involving only the unknowns x_{i-1} and x_{i+1}

$$S'_{i-1}x_{i-1} + R'_i x_{i+1} = f'_i. \tag{2}$$

Since the columns overlapped by the $2m \times m$ matrix $\begin{pmatrix} R_{i-1} \\ S_i \end{pmatrix}$ are linearly independent, we use a partial pivoting LU factorization

$$P_i \begin{pmatrix} R_{i-1} \\ S_i \end{pmatrix} = \begin{pmatrix} L_i \\ T_i \end{pmatrix} U_i = \begin{pmatrix} L_i & \\ T_i & I \end{pmatrix} \begin{pmatrix} U_i \\ 0 \end{pmatrix} \tag{3}$$

and premultiply (1) by the permutation matrix P_i and the inverse of the lower triangular matrix in (3) to obtain

$$\begin{pmatrix} L_i & \\ T_i & I \end{pmatrix}^{-1} P_i \begin{pmatrix} S_{i-1} & R_{i-1} & \\ & S_i & R_i \end{pmatrix} \equiv \begin{pmatrix} \tilde{S}_{i-1} & U_i & \tilde{R}_i \\ S'_{i-1} & & R'_i \end{pmatrix}, \begin{pmatrix} L_i & \\ T_i & I \end{pmatrix}^{-1} P_i \begin{pmatrix} f_{i-1} \\ f_i \end{pmatrix} \equiv \begin{pmatrix} \tilde{f}_i \\ f'_i \end{pmatrix}, \tag{4}$$

where S'_{i-1}, R'_i and f'_i are the blocks of the reduced equation (2). After k steps of reduction, the coefficient matrix obtained is of size $\lceil N/s \rceil + 1$ where $s = 2^k$, and it can be further on reduced by combining two successive block row equations (for each $i = (2j-1)s + 1$, with $j = 1, 2, \ldots, \lfloor N/(2s) \rfloor$)

$$\begin{pmatrix} S^{(k)}_{i-s} & R^{(k)}_{i-1} & & \\ & S^{(k)}_i & R^{(k)}_{i+s-1} \end{pmatrix} \begin{pmatrix} x_{i-s} \\ x_i \\ x_{i+s} \end{pmatrix} = \begin{pmatrix} f^{(k)}_{i-1} \\ f^{(k)}_{i+s-1} \end{pmatrix}$$

to give $S^{(k+1)}_{i-s} x_{i-s} + R^{(k+1)}_{i+s-1} x_{i+s} = f^{(k+1)}_{i+s-1}$. The reduction ends after $p = \lceil \log_2 N \rceil$ steps when the 2×2 block linear system

$$\begin{pmatrix} B_a & B_b \\ S^{(p)}_1 & R^{(p)}_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_{N+1} \end{pmatrix} = \begin{pmatrix} d \\ f^{(p)}_N \end{pmatrix} \tag{5}$$

is obtained. The algorithm proceeds with the solution of (5) and the back-substitution phase where the unknowns $x_2, \ldots, x_N$ are computed.

The presence of null blocks in the equations (4) allows us to reduce the memory requirement and the number of computations. In order to complete the reduction, we save, after $k+1$ steps of reduction, the matrices $S^{(k+1)}_{i-s}$ and $R^{(k+1)}_{i+s-1}$ in

place of $S_{i-s}^{(k)}$ and $R_{i+s-1}^{(k)}$ respectively, the product $T_i L_i^{-1}$ in place of $S_i^{(k)}$ and the vector $f_{i+s-1}^{(k+1)}$ in place of $f_{i+s-1}^{(k)}$. Moreover, for the back-substitution phase, we save the first m elements of $P_i^{(k)} \begin{pmatrix} f_{i-1}^{(k)} \\ f_{i+s-1}^{(k)} \end{pmatrix}$ in place of $f_{i-1}^{(k)}$. The extra memory requirement is $m \times m$ for each of the $N-1$ reductions and corresponds to the first m rows of $P_i^{(k)} \begin{pmatrix} S_{i-s}^{(k)} \\ R_{i+s-1}^{(k)} \end{pmatrix}$ used in the back-substitution phase. The computational cost of the factorization is $\frac{14}{3}m^3N$ and of the back-substitution phase is $6m^2N$ to leading order in powers of m and N. Since $N \gg m$ and the *BABDCR* algorithm can reduce successive pairs of block row equations independently, the algorithm can be efficiently parallelized.

3 BVPs and the *MIRKDC* code

The code *MIRKDC* [10] solves BVPs

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)), \qquad t \in [a, b] \tag{6}$$

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

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \begin{pmatrix} \mathbf{g}_0(\mathbf{y}(a)) \\ \mathbf{g}_1(\mathbf{y}(b)) \end{pmatrix} = 0. \tag{7}$$

It uses Mono-Implicit Runge Kutta (MIRK) formulae to discretize (6) on a given subdivision $\{t_i\}_{i=0}^N$ of $[a, b]$. A continuous solution approximation is obtained using a Continuous MIRK (CMIRK) scheme, to provide defect control and mesh selection capabilities. The MIRK scheme applied to the BVP system (6)-(7) on N subintervals, yields the nonlinear system

$$\mathbf{\Phi}(\mathbf{Y}) = (\mathbf{\Phi}_0(\mathbf{Y})^T, \ldots, \mathbf{\Phi}_N(\mathbf{Y})^T)^T = 0, \quad \text{where } \mathbf{\Phi}_i : \mathbb{R}^{m(N+1)} \longrightarrow \mathbb{R}^m$$

and $\mathbf{Y} = (\mathbf{y}_0^T, \ldots, \mathbf{y}_N^T)^T$, $\mathbf{y}_j \in \mathbb{R}^m$, which is solved using the Newton iteration $\mathbf{Y}^{(q+1)} = \mathbf{Y}^{(q)} + \Delta\mathbf{Y}^{(q)}$, for $q = 0, 1, \ldots$, where

$$\left[\frac{\partial \mathbf{\Phi}(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}}\right] \Delta\mathbf{Y}^{(q)} = -\mathbf{\Phi}(\mathbf{Y}^{(q)}) \tag{8}$$

given $\mathbf{Y}^{(0)}$. For the separated boundary conditions (7) these linear systems have an ABD structure as in Fig. 1 with

$$S_i = -I - h_i K_{i,i}, \qquad R_i = I - h_i K_{i+1,i},$$

where blocks $K_{i,j}$ depend on the used Runge-Kutta formulae.

MIRKDC uses the following variable names: `neqns` and `Nsub` are, respectively, the order m of the system (6) and the number N of subintervals of the current mesh, `leftbc` is the number of boundary conditions at the point a and `MxNsub` is the user defined maximum number of subintervals of $[a, b]$. It employes *COLROW* to solve systems (8).

We have written a variant of MIRKDC, called *MIRKDC_NOSEP*, which solves the system of BVPs (6) with general non-separated boundary conditions. The algorithm uses the discretizations of *MIRKDC* resulting in linear systems (8) with the BABD form, as in Fig. 2, that are solved using the BABD solver *BABDCR*. Therefore, *MIRKDC_NOSEP* essentially replaces *COLROW* with *BABDCR*. To do this, we make some modifications inside the *MIRKDC* code:

- the permutation vector array, of length `neqns*(MxNsub+1)` is replaced by a vector array of length `2*neqns*MxNsub`
- the fill-in described in section 2, adds `(Nsub-1)*(neqns**2)` locations in the array `blocks` which contain also the blocks S_i, R_i, as represented in Fig. 2, of the current BABD Jacobian in (8)
- the arrays `top` and `bop` contain the blocks B_a and B_b of the Jacobian in (8), represented in Fig. 2, which are of dimension `neqns`×`neqns` instead of `leftbc`×`neqns` and `(neqns-leftbc)`×`neqns`, respectively
- in *BABDCR* the right hand side associated with the linear system is overwritten by the solution, while *COLROW* doesn't overwrite the solution. Therefore, before the call to *BABDCR* the right hand side occupies the same locations as for the solution

4 Comparisons among the linear system solvers

We compare *BABDCR* with *COLROW* and *RSCALE* on the ABD and BABD linear systems generated by the codes *MIRKDC* and *MIRKDC_NOSEP*. First, we discuss the ABD linear systems generated by *MIRKDC* applied to a linear BVP $\mathbf{y}' = M\mathbf{y}(t)$ with $M \in \mathbb{R}^{m\times m}$ and linear boundary conditions $B_{top}\mathbf{y}(a) = \mathbf{d}_a \in \mathbb{R}^{m_0}$ and $B_{bot}\mathbf{y}(b) = \mathbf{d}_b \in \mathbb{R}^{m-m_0}$. Here we fix $m = 20$ and $m_0 = 10$. Both B_{top}, B_{bot} and M are randomly generated full matrices and such that the BVP is well-conditioned. *BABDCR* and *RSCALE* requires the simple

transformation

$$B_a = \begin{pmatrix} B_{top} \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times m},\ B_b = \begin{pmatrix} 0 \\ B_{bot} \end{pmatrix} \in \mathbb{R}^{m\times m} \text{ and } d = \begin{pmatrix} d_a \\ d_b \end{pmatrix} \in \mathbb{R}^m\,,$$

in order to be applied to a system with the BABD structure in Fig. 2. The timings and errors, in Table 1, lead us to prefer *COLROW* over the other algorithms. Indeed, *COLROW* is more than 2 times faster than *BABDCR* and more than 4 times faster than *RSCALE*. Moreover, the errors for *COLROW* and *BABDCR* are similar, but *RSCALE* is less accurate. These results essentially agree with the theoretical computational costs of the three solvers. In fact, applied to the ABD linear system in Fig. 1, to leading order in m, *COLROW* requires $\left(\frac{5}{3}m^3 + mm_0^2\right)N$ operations and *RSCALE* $\left(\frac{20}{3}m^3\right)N$.

Table 1
ABD systems generated by MIRKDC applied to a linear problem.

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

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

(1) M is a well-conditioned matrix with eigenvalues -102, -10, -7, -4, -3, -2.5, -1.3, -1, -0.5, -0.4, 0.2, 0.3, 1, 1, 2, 2.5, 3, 4, 11, 25;
(2) M has eigenvalues -9, -3.5, -3, -2, -2, -1.5, -1.5, -1.25, -0.5, -1e-08, 0.25, 0.5, 0.5 , 1, 3, 4, 5, 7, 8, 1e+08.

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

Fig. 3. ABD system of doubled size equivalent to the BABD system in Fig. 2. z_i, $i = 1, \ldots, N+1$ are the new unknowns, $z_{N+1} = z_N = \ldots = z_1 = x_1$.

$$\begin{pmatrix} -I & I & & & & & & \\ S_1 & 0 & R_1 & & & & & \\ & -I & 0 & I & & & & \\ & & & \ddots & & & & \\ & & & S_N & 0 & R_N & & \\ & & & & -I & 0 & I & \\ & & & & & B_b & B_a & \end{pmatrix} \begin{pmatrix} x_1 \\ z_1 \\ x_2 \\ z_2 \\ \vdots \\ x_{N+1} \\ z_{N+1} \end{pmatrix} = \begin{pmatrix} 0 \\ f_1 \\ 0 \\ \vdots \\ f_N \\ 0 \\ d \end{pmatrix}$$

more accurate than *RSCALE*.

Table 2
Times for the solution of BABD systems generated by MIRKDC_NOSEP applied to a linear 20×20 BVP.

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

Table 3
Errors for the solution of BABD systems generated by MIRKDC_NOSEP applied to a linear 20×20 BVP.

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

In order to better emphasize the advantages of using *BABDCR*, Tables 4-5 give statistics for calls to *MIRKDC_NOSEP* (that uses *BABDCR*) and to *MIRKDC* applied to the BVP of double size $\mathbf{y}' = M\mathbf{y}(t)$, $\mathbf{z}' = \mathbf{0}$ with separated boundary conditions $\mathbf{y}(a) - \mathbf{z}(a) = \mathbf{0}$ and $B_a\mathbf{z}(b) + B_b\mathbf{y}(b) = \mathbf{d}$. Both codes are applied to the problem with M having eigenvalues as in case (1) using the MIRK/CMIRK scheme of order 4 [10]. From Tables 4 and 5, *BABDCR* in *MIRKDC_NOSEP* saves more than one half of the linear algebra time with respect to using *COLROW*. Though this result is valid for the problem considered, results in Table 2 show that, for a general BVP with non-separated boundary conditions, *BABDCR* in *MIRKDC* runs faster than *COLROW*.

Table 4
MIRKDC_NOSEP (using *BABDCR*) for the linear problem in case (1) with an initial mesh of 256 points and tolerance 1e-07

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

Table 5
MIRKDC (using *COLROW*) for the linear problem of double size in case (1) with an initial mesh of 256 points and tolerance 1e-07.

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

5 Parallel implementation of *BABDCR*

We consider a distributed memory parallel implementation of *BABDCR* called *PARABABDCR*. This implementation has good speed-up when applied to a random BABD linear system, even on 8 processors (see Table 6). So, *BABDCR* could give a faster *MIRKDC* package on distributed parallel architectures than *PMIRKDC* [14] using *RSCALE*. Direct comparisons with *PMIRKDC* are not possible since it is for shared memory architectures.

Table 6
Speedup for the PARABABDCR *algorithm with N=1024.*

	NPROCS=2	NPROCS=4	NPROCS=8
m=64	1.976	3.473	5.977
m=16	1.529	2.806	3.343
m=4	1.227	1.424	1.057

References

[1] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut and M. Paprzycki, Almost block diagonal linear systems: sequential and parallel solution techniques, and applications, *Numer. Linear Algebra Appl.*, **7** (2000), 275-317.

[2] P. Amodio and M. Paprzycki, A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comp* **18** 1 (1997), 56-68.

[3] P. Amodio and G. Romanazzi, BABDCR: a Fortran 90 package for the solution of Bordered ABD systems, Technical Report n. 40/04, Dipartimento di Matematica, Università di Bari, 2004 (submitted).

[4] U.M. Ascher, J. Christiansen and R.D. Russell, Algorithm 569: COLSYS: Collocation Software for Boundary-Value ODEs, *ACM Trans. Math. Softw.*, **7** 2 (1981) 223-229.

[5] G. Bader and U. Ascher, A new basis implementation for a mixed order boundary value ODE solver, *SIAM J. Sci. Statist. Comput.* **8** 4 (1987) 483-500.

[6] J.R. Cash, G. Moore and R.W. Wright, An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems, *ACM Trans. Math. Software*, **27** 2 (2001) 245-266.

[7] J.R. Cash and R.W. Wright, A deferred correction method for nonlinear two-point boundary value problems: Implementations and numerical evaluation, *SIAM J. Sci. Statist. Comp.*, **12** 4 (1991) 971-989.

[8] C. De Boor and R. Weiss, SOLVEBLOK: A package for solving almost block diagonal linear systems, *ACM Trans. Math. Softw.* **6** 1 (1980) 80-87.

[9] J.C. Diaz, G. Fairweather and P. Keast, FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination., *ACM Transactions on Mathematical Software* **9** 3 (1983) 358-375.

[10] W.H. Enright and P.H. Muir, Runge-Kutta software with defect control for boundary value ODEs, *SIAM J. Sci. Comput.* **17** 2 (1996) 479-497.

[11] G. Fairweather and I. Gladwell, Algorithms for Almost Block Diagonal Linear Systems, *SIAM Review* **46** 1 (2004) 49-58.

[12] B. Garrett and I. Gladwell, Solving Bordered Almost Block Diagonal Systems Stably and Efficiently, *J. Comput. Methods Sci. Engrg.*, **1** (2001) 75-98.

[13] K.R. Jackson and R.N. Pancer, The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs, Techinical Report No. 255/91, Computer Science Department, University of Toronto, 1992.

[14] P.H. Muir, R.N. Pancer and K.R. Jackson, PMIRKDC: a parallel mono-implicit Runge-Kutta code with defect control for boundary value ODEs, *Paral. Comput.* **29** (2003) 711-741.

[15] R.W. Wright, J. Cash and G. Moore, Mesh selection for stiff two-point boundary value problems, *Numer. Alg.* **7** (1994) 205-224.

[16] J.M. Varah, Alternate row and column elimination for solving certain linear systems, *SIAM J. Numer. Anal.* **13** (1976) 71-75.