Algorithm 859: BABDCR—A Fortran 90 Package for the Solution of Bordered ABD Linear Systems

PIERLUIGI AMODIO and GIUSEPPE ROMANAZZI
Università di Bari, Italy

BABDCR is a package of Fortran 90 subroutines for the solution of linear systems with bordered almost block diagonal coefficient matrices. It is designed to handle matrices with blocks of the same size, that is, having a block upper bidiagonal structure with an additional block in the right upper corner. The algorithm implemented in the package performs cyclic reduction of the coefficient matrix in order to reduce the fill-in due to the corner block.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Sparse, structured, and very large systems (direct and iterative methods)*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Algorithm design and analysis*; *documentation*

General Terms: Algorithms, Documentation

Additional Key Words and Phrases: Linear systems, bordered almost block diagonal matrices, numerical solution, cyclic reduction

1. INTRODUCTION

Almost block diagonal (ABD) matrices are sparse matrices characterized by a very special sparsity pattern: The nonzero elements are grouped in block rows; there is no intersection between the nonzero columns of two nonconsecutive block rows; finally, the main diagonal entries always lie in the nonzero blocks. Bordered ABD (BABD) matrices satisfy a further property: The first (or the last) block row has an additional block in the right upper (left lower) corner whose nonzero columns intersect only the nonzero columns of the last (first) block row.

In this article we are interested in BABD linear systems

$$Ax = f \tag{1}$$

with the coefficient matrix A having the special BABD structure shown in

Authors' address: P. Amodio and G. Romanazzi, Dipartimento di Matematica, Università di Bari, Bari, Italy; email: {amodio; romanazzi}@dm.uniba.it.

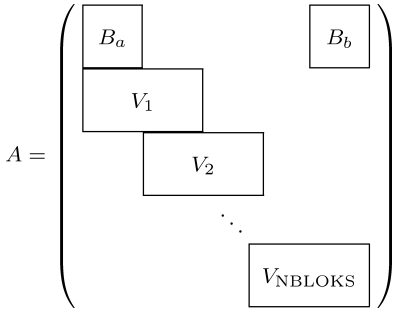

Fig. 1. Structure of the BABD matrix.

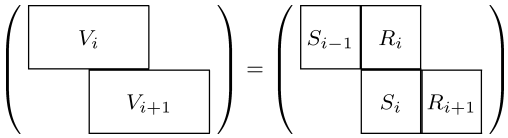

Fig. 2. Successive blocks in the BABD coefficient matrix.

Figure 1; the blocks V_i all have the same size and overlap (the number of columns shared by two successive blocks) as predecessor and successor blocks.

Specifically, using the same notation as in Diaz et al. [1983a, 1983b], we require that each block V_i, for $i = 1, \ldots,$ NBLOKS, has NRWBLK rows and $2 *$ NRWBLK columns, while B_a and B_b (called boundary blocks) are square blocks of dimension NRWBLK. Moreover, in this structure the overlap between two adjacent blocks is always NRWBLK. This means that for any index i, the blocks V_i and V_{i+1} can be represented as in Figure 2, where S_{i-1}, R_i, S_i and R_{i+1} all have size NRWBLK by NRWBLK.

This kind of linear system arises in a large variety of contexts, particularly in the discretization of boundary value problems (BVPs) for ordinary and partial differential equations (see, e.g., Ascher et al. [1995]). The presence of the boundary blocks is due to the occurrence of nonseparated boundary conditions.

Because of its relevance in BVP codes, the solution of ABD systems has been the subject of long-term research, and several codes have been proposed. For a survey on the topic, we refer to Amodio et al. [2000] and the more recent article of Fairweather and Gladwell [2004]. The first code to be applied to ABD systems is SOLVEBLOK in De Boor and Weiss [1980]; it requires fill-in to ensure stability, as in the case of standard LU factorization applied to banded systems. In Varah [1976] the sparsity structure of ABD matrices is exploited in order to perform the alternate row and column elimination and thus avoid fill-in. The algorithm in Varah [1976] is based on Lam's method [1974] in which alternate row and column interchanges are used to avoid fill-in, but only row elimination is performed throughout. The packages COLROW and ARCECO in Diaz et al. [1983a, 1983b], as well as the NAG routine F01LHF in Brankin and Gladwell [1990], are based on Varah's algorithm. The same algorithm has been applied to the solution of ABD systems with additional zeroes inside each block. For example, ABDPACK in Majaess et al. [1992a, 1992b] implements an

alternate row and column elimination that exploits the sparsity structure due to the approximation of BVPs by means of spline collocation.

Due to the difficulty of treating boundary blocks, the solution of BABD systems has received much less consideration. BABD systems may be rewritten as ABD systems in order to use ABD packages, but this approach requires doubling the size of the system. On the other hand, the boundary block outside the ABD structure necessarily implies fill-in. To the best of our knowledge, the only routines that can handle BABD systems are DECOMP/SOLVE (for the factorization and solution, respectively) inside the PASVAR code in Lentini and Pereyra [1977], which is designed to solve BVPs for ODEs. These two routines use an alternate row and column elimination with fill-in within the bottom block row associated with the presence of the nonseparated boundary conditions. Alternative approaches have been specifically designed for parallel implementation. We recall the SLU and SQR algorithms in Wright [1992, 1994] and the RSCALE routine inside the PMIRKDC package of Muir et al. [2003], which is designed to solve BVPs for ODEs.

In this article we describe the use of a new sequential package, BABDCR, for the solution of BABD systems. It is based on the cyclic reduction algorithm originally proposed in Amodio and Paprzycki [1997] that was implemented on a parallel computer. It is well-known that cyclic reduction applied to block tridiagonal (or ABD) systems is extremely competitive only on parallel computers. However, the presence of boundary blocks implies fill-in and a higher computational cost in all sequential algorithms (e.g., the LU factorization). Therefore, we show that the cyclic reduction algorithm can also be effective on a sequential computer.

In Sections 2 and 3 we summarize this cyclic reduction approach and evaluate its computational cost. Then, in Section 4 we describe the software. Our package is written in Fortran 90 and available in double precision. It has been tested on an AlphaServer DS20E with a 667 MHz EV67 processor with a Compaq Fortran 90 (formerly Digital Fortran 90) compiler. Finally, in Section 5 we compare the BABDCR code with other available codes.

2. THE BABDCR ALGORITHM

The basic idea in developing this code (see Amodio and Paprzycki [1997]) is to reduce Eq. (1) cyclically in order to derive systems of lower dimension (involving less unknowns) with the same BABD structure. Let us rewrite system (1) as

$$\begin{pmatrix} B_a & & & & & B_b \\ S_0 & R_1 & & & & \\ & S_1 & R_2 & & & \\ & & \ddots & \ddots & & \\ & & & S_{\mathrm{NBLOKS}-1} & R_{\mathrm{NBLOKS}} & \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{\mathrm{NBLOKS}-1} \\ x_{\mathrm{NBLOKS}} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{\mathrm{NBLOKS}-1} \\ f_{\mathrm{NBLOKS}} \end{pmatrix}, \quad (2)$$

where each block in the BABD matrix has dimension NRWBLK by NRWBLK. Suppose for the moment that NBLOKS $= 2^p$. Since operations involving the boundary block B_b would create fill-in, all the reduced systems always keep

x_0 and x_{NBLOKS} among the unknowns, and leave unchanged the first row of Eq. (2). For this reason, the first reduction produces the following linear system of dimension (NBLOKS/2 + 1)

$$\begin{pmatrix} B_a & & & & B_b \\ S'_0 & R'_2 & & & \\ & S'_2 & R'_4 & & \\ & & \ddots & \ddots & \\ & & & S'_{\mathrm{NBLOKS}-2} & R'_{\mathrm{NBLOKS}} \end{pmatrix} \begin{pmatrix} x_0 \\ x_2 \\ \vdots \\ x_{\mathrm{NBLOKS}-2} \\ x_{\mathrm{NBLOKS}} \end{pmatrix} \begin{pmatrix} f_0 \\ f'_2 \\ \vdots \\ f'_{\mathrm{NBLOKS}-2} \\ f'_{\mathrm{NBLOKS}} \end{pmatrix}.$$

There are several ways to derive the aforementioned reduced system from Eq. (2). But, since the original matrix is essentially block bidiagonal (the first block row is not modified), each block row of the reduced system may be obtained by two consecutive rows in Eq. (2). That is, for i odd, the subsystem

$$\begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} \begin{pmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{pmatrix} = \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix} \tag{3}$$

is transformed in order to obtain one row of the reduced system involving only the unknowns x_{i-1} and x_{i+1}

$$S'_{i-1}x_{i-1} + R'_{i+1}x_{i+1} = f'_{i+1}. \tag{4}$$

If R_i is nonsingular, this could be achieved by multiplying Eq. (3) on the left by

$$\begin{pmatrix} I \\ -S_i R_i^{-1} & I \end{pmatrix}, \tag{5}$$

thus obtaining Eq. (4) with $S'_{i-1} = -S_i R_i^{-1} S_{i-1}$, $R'_{i+1} = R_{i+1}$ and $f'_{i+1} = f_{i+1} - S_i R_i^{-1} f_i$.

However, this procedure might be unstable (see the next section). For this reason, we perform a partial pivoting LU factorization of the 2 ∗ NRWBLK by NRWBLK overlap columns:

$$P_i \begin{pmatrix} R_i \\ S_i \end{pmatrix} = \begin{pmatrix} L_i \\ T_i \end{pmatrix} U_i = \begin{pmatrix} I \\ G_i \end{pmatrix} L_i U_i = \begin{pmatrix} I & \\ G_i & I \end{pmatrix} \begin{pmatrix} L_i U_i \\ O \end{pmatrix}, \tag{6}$$

($G_i = T_i L_i^{-1}$), where P_i is a 2 ∗ NRWBLK by 2 ∗ NRWBLK permutation matrix obtained by the NRWBLK row interchanges. Then,

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} = \begin{pmatrix} \widehat{S}_{i-1} & L_i U_i & \widehat{R}_{i+1} \\ S'_{i-1} & & R'_{i+1} \end{pmatrix} \tag{7}$$

and

$$\begin{pmatrix} I & \\ -G_i & I \end{pmatrix} P_i \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix} = \begin{pmatrix} \widehat{f}_i \\ f'_{i+1} \end{pmatrix}. \tag{8}$$

Taking into account the second row in Eqs. (7) and (8) yields Eq. (4) of the reduced system. The first row of Eqs. (7) and (8) are used in the back-substitution

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

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

$$\begin{pmatrix} S_{i-s}^{(k)} & R_i^{(k)} & \\ & S_i^{(k)} & R_{i+s}^{(k)} \end{pmatrix} \begin{pmatrix} x_{i-s} \\ x_i \\ x_{i+s} \end{pmatrix} = \begin{pmatrix} f_i^{(k)} \\ f_{i+s}^{(k)} \end{pmatrix} \tag{9}$$

so as to obtain

$$S_{i-s}^{(k+1)} x_{i-s} + R_{i+s}^{(k+1)} x_{i+s} = f_{i+s}^{(k+1)}. \tag{10}$$

In analogy with Eqs. (7) and (8), from Eq. (9) we also deduce the following equality

$$\widehat{S}_{i-s}^{(k)} x_{i-s} + L_i U_i x_i + \widehat{R}_{i+s}^{(k)} x_{i+s} = \widehat{f}_i^{(k)}, \tag{11}$$

which is used in the back-substitution phase to compute x_i from x_{i-s} and x_{i+s}.

The reduction ends after p steps, when the two by two block linear system

$$\begin{pmatrix} B_a & B_b \\ S_0^{(p)} & R_{\text{NBLOKS}}^{(p)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_{\text{NBLOKS}} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_{\text{NBLOKS}}^{(p)} \end{pmatrix} \tag{12}$$

is obtained. The algorithm proceeds with the solution of system (12) and the back-substitution phase where the unknowns $x_1, \ldots, x_{\text{NBLOKS}-1}$ are computed.

If NBLOKS is not a power of two, the algorithm behaves in an essentially similar manner. The first block row is never modified until, after $p = \lceil \log_2(\text{NBLOKS}) \rceil$ steps, system (12) is obtained. After the k-th step of reduction is performed, the dimension of the reduced system is $\lceil \text{NBLOKS}/s \rceil + 1$, where $s = 2^k$ and $\lfloor \lceil \text{NBLOKS}/s \rceil / 2 \rfloor$ reductions from Eqs. (9) to (10) are required to derive the new reduced matrix. Since the first and last unknowns need not be affected by the reduction process, if the dimension of the reduced matrix (including the row with boundary blocks) is even, then also the last block row remains unchanged in the reduction. Thus, for example, if NBLOKS is odd, then after one step of reduction, the dimension of the first reduced block matrix is ((NBLOKS + 1)/2 + 1) and the unknowns of the first reduced system are $x_0, x_2, x_4, \ldots, x_{\text{NBLOKS}-1}, x_{\text{NBLOKS}}$.

3. COMPUTATIONAL COST AND STABILITY

We measure the computational cost of the BABDCR algorithm in terms of total number of *flops* (each flop represents one of the four arithmetic floating point operations) and required memory. In Section 5 we will compare the values we obtain with those given by the existing codes when applied to BABD systems.

The factorization of the coefficient matrix (i.e., one call to the subroutine BABDCR_FACT described next) requires NBLOKS − 1 reductions from Eq. (9) to Eq. (10) and the partial pivoting LU factorization of the two by two block matrix in system (12). The cost of each reduction is approximately $\frac{14}{3} * \text{NRWBLK}^3$ flops, and that of the two by two block factorization is approximately $\frac{16}{3} * \text{NRWBLK}^3$ flops. Thus, the factorization requires $\frac{14}{3} * \text{NRWBLK}^3 *$ NBLOKS flops to leading order in powers of NBLOKS and NRWBLK.

Solving the linear system (2) when the matrix is already factorized (i.e., one call to the subroutine BABDCR_SOLV described next) requires NBLOKS − 1 reductions of the righthand-side from Eq. (9) to Eq. (10), NBLOKS − 1 back-substitutions in Eq. (11) to compute the unknown x_s from x_{i-s} and x_{i+s} (previously determined), and the solution of the two by two block linear system (12) previously factorized. The cost of each reduction is approximately $2 * \text{NRWBLK}^2$ flops, that of each back-substitution step is $4 * \text{NRWBLK}^2$, and of the solution of the two by two block system is $8 * \text{NRWBLK}^2$ flops. Therefore, the solution of a linear system with the BABDCR algorithm requires approximately $6 * \text{NRWBLK}^2 * \text{NBLOKS}$ flops.

The flop component of the cost of the BABDCR algorithm does not change if the factorization and solution steps are performed together. However, the memory component of the cost changes drastically. If the reduction phase of the coefficient matrix and the righthand-side are carried out at the same time (this is obtained by one call to the subroutine BABDCR_FACTSOLV described next), then the relevant part of the factorization of the matrix is stored in place of the original coefficient matrix and the algorithm does not require fill-in. If we want to compute the factorization only once and solve several linear systems, then we need to store the permutation matrices and all the computed blocks G_i, as well. This means that the algorithm requires a fill-in array of size $\text{NRWBLK}^2 *$ (NBLOKS − 1) (which is almost one-half of the memory required to store the coefficient matrix) and an integer array of size $2 * \text{NRWBLK} * \text{NBLOKS}$.

We now discuss the stability of our algorithm in the special case $S_i = S$ and $R_i = R$ for each i. Such a situation arises, for example, when an autonomous ODE-BVP is discretized by using constant stepsize. Let λ and x be one eigenvalue and the corresponding eigenvector of the matrix pencil (S, R). Then from Eqs. (6) and (7) (the indices i have been neglected because of the constant blocks), we have

$$\begin{aligned} S'x &= \left(\begin{array}{cc} -G & I \end{array}\right) P \left(\begin{array}{c} S \\ 0 \end{array}\right) x = \lambda \left(\begin{array}{cc} -G & I \end{array}\right) P \left(\begin{array}{c} R \\ 0 \end{array}\right) x = -\lambda \left(\begin{array}{cc} -G & I \end{array}\right) P \left(\begin{array}{c} 0 \\ S \end{array}\right) x \\ &= -\lambda^2 \left(\begin{array}{cc} -G & I \end{array}\right) P \left(\begin{array}{c} 0 \\ R \end{array}\right) x = -\lambda^2 R'x, \end{aligned}$$

where we have used $\left(\begin{array}{c} R \\ 0 \end{array}\right) = \left(\begin{array}{c} R \\ S \end{array}\right) - \left(\begin{array}{c} 0 \\ S \end{array}\right)$ and $\left(\begin{array}{cc} -G & I \end{array}\right) P \left(\begin{array}{c} R \\ S \end{array}\right) = 0$.

This means that as the reduction process goes on, the eigenvalues with modulus less than 1 approach 0 and the eigenvalues with modulus greater than 1 approach infinity. Hence, the reduction by means of matrix (5) may be unstable because there is no control on the growth (in modulus) of the elements in $S^{(k)}$ (obtained after k steps of reduction). Conversely, the BABDCR algorithm

should be stable because all the elements of both T_i and L_i have modulus less or equal to 1. Therefore, we expect that the rows of $G_i^{(k)}$ go quickly to zero, and the matrices $S^{(k)}$ and $R^{(k)}$ converge to S^* and R^* with the property that if one row in S^* is nonzero, then the corresponding one in R^* is zero (see Section 5). Due to the use of the partial pivoting LU factorization, "pathological" cases may occur; however, in such cases the growth of the elements of S' and R' depends on NRWBLK $\ll$ NBLOKS and is restricted only to one step of reduction.

4. DESCRIPTION OF THE SOFTWARE

The BABDCR package has four main subroutines:

—BABDCR_FACT factorizes the BABD coefficient matrix in Eq. (2);

—BABDCR_SOLV solves the linear system (2) with a coefficient matrix factorized by BABDCR_FACT;

—BABDCR_SOLVT solves the linear system (2) with the transposed of the coefficient matrix factorized by BABDCR_FACT; and

—BABDCR_FACTSOLV factorizes the BABD coefficient matrix, and at the same time solves the linear system (2).

The last subroutine is convenient if only one BABD linear system needs to be solved. The original coefficient matrix is replaced with part of the factorization and cannot be used any longer. On the other hand, the first two subroutines (contained in the module BABDCR) solve system (2) in two successive steps: BABDCR_SOLV uses the output of BABDCR_FACT to compute the solution of system (2) (the arrays containing the cyclic reduction factorization are not modified by successive calls to BABDCR_SOLV). Thus, the solution of p linear systems with the same coefficient matrix can be computed by means of one call to BABDCR_FACT followed by p calls to BABDCR_SOLV. This procedure yields a great decrease in the number of operations compared to the multiple use of BABDCR_FACTSOLV; however, it requires fill-in vectors (see the following). Finally, if A is factorized with BABDCR_FACT, then BABDCR_SOLVT uses its output to solve the linear system $A^T z = f$. This subroutine is essentially used to compute the one-norm of the inverse of the coefficient matrix by means of the subroutine DONEST in Higham [1989] that, in order to evaluate the one-norm of a square, double precision matrix B, requires the products Bx and $B^T x$ for some given vectors x.

We include in our package a driver program to link the four subroutines. Specifically, it allocates storage, reads the input dataset, and calls the subroutines.

The package requires that the coefficient matrix in input is given as in Figure 3, that is, the blocks $V_i = (S_{i-1}, R_i)$ must be given sequentially in a NRWBLK by NRWBLK by $2 *$ NBLOKS three-dimensional array MATR_A; boundary blocks are saved in two NRWBLK by NRWBLK arrays LFTBLK and RGTBLK. The righthand-side f must be assigned in a vector of length NRWBLK $*$ (NBLOKS $+ 1$) and is stored in a NRWBLK by NBLOKS$+1$ array.

Header comments in each procedure provide details regarding the specification of input and output parameters and the workspace requirements. The

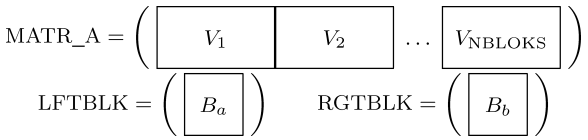

Fig. 3. Structure of the input coefficient matrix.

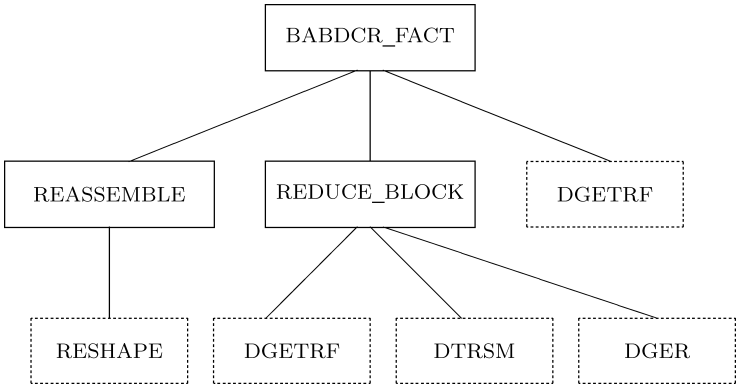

Fig. 4. BABDCR_FACT subroutine hierarchy.

structure of BABDCR_FACT and BABDCR_SOLV is represented in Figures 4 and 5. Dashed blocks contain the Fortran 90 intrinsic procedure RESHAPE, the BLAS routines DGER, DTRSM, DGEMV, and DDOT, and the Lapack routines DGETRF and DGETRS.

In BABDCR_FACT there are calls to REASSEMBLE and REDUCE_BLOCK. The subroutine REASSEMBLE allows the assembly of a $2*$NRWBLK by NRWBLK block from two consecutive blocks of size NRWBLK by NRWBLK. The output obtained is used by REDUCE_BLOCK, which applies one step of reduction to obtain one block row from two consecutive ones (see Eqs. (9) and (10)). The operations performed in REDUCE_BLOCK are summarized in Figure 6, where the variables considered have the same meaning as those in Section 2.

Since the reduction is applied only to the coefficient matrix (not to the righthand-side), we need to save the block G_i which is used in REDUCE_RHS; moreover, the matrices L_i and U_i of the LU factorization of the $2*$NRWBLK by NRWBLK block $\begin{pmatrix} R_i^{(k)} \\ S_i^{(k)} \end{pmatrix}$ (computed by means of the LAPACK routine DGETRF) and the fill-in block F_i of size NRWBLK by NRWBLK containing the nonzero rows of $\widehat{S}_{i-s}^{(k)}$ and $\widehat{R}_{i+s}^{(k)}$, which are used in SOLVE_BLOCK. Finally, $S_{i-s}^{(k)}$ and $R_{i+s}^{(k)}$ are replaced by the computed blocks of the new reduced matrix, namely, $S_{i-s}^{(k+1)}$ and $R_{i+s}^{(k+1)}$.

On exit, BABDCR_FACT outputs two new arrays containing the fill-in blocks F_i and the permutations, saved in a NRWBLK by NRWBLK by NBLOKS $-$ 1 array and a $2 *$ NRWBLK by NBLOKS array, respectively. The arrays containing the coefficient matrix now have the structure shown in Figure 7: the first and last blocks of MATR_A and the boundary blocks contain the LU factorization of the matrix in Eq. (12) (see Figure 8), whereas each LUG_i block contains

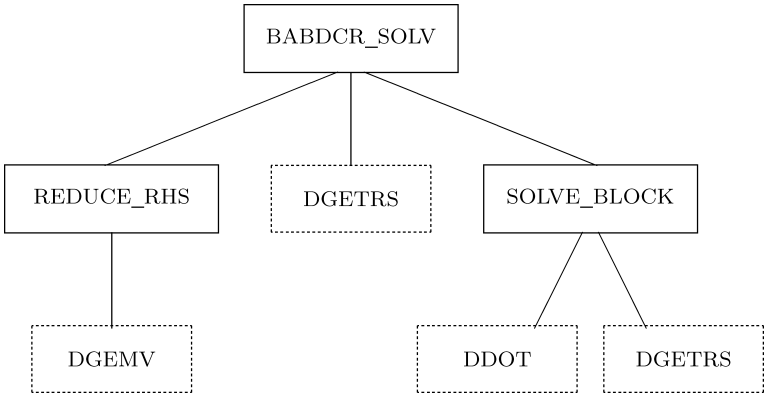

Fig. 5. BABDCR_SOLV subroutine hierarchy.

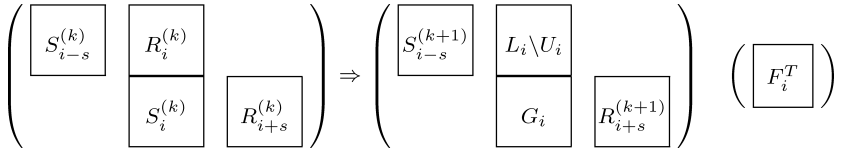

Fig. 6. Operations performed in REDUCE_BLOCK.

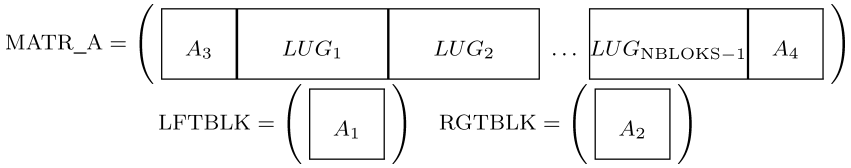

Fig. 7. Structure of the coefficient matrix on exit from BABDCR_FACT.

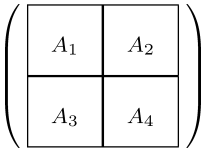

Fig. 8. Structure of the LU factorization of Eq. (12).

L_i, U_i, and G_i (saved as in Figure 6) reshaped as a $2 * \mathrm{NRWBLK}$ by NRWBLK array.

The subroutine BABDCR_SOLV uses the subroutines REDUCE_RHS to compute the righthand-side of Eq. (10) from Eq. (9), and SOLVE_BLOCK to go back, that is, to compute x_s from Eq. (11) (x_{i-s} and x_{i+s} are known values). The LAPACK routine DGETRS performs the solution of the last two by two block system factorized with DGETRF.

Finally, the subroutine BABDCR_FACTSOLV makes use of the subroutines REDUCE (which include the subroutines REDUCE_BLOCK and REDUCE_RHS), REASSEMBLE, and SOLVE_BLOCK. Since the linear

system is solved step-by-step with the factorization, the block G_i is used as soon as it is computed and so does not need to be saved. For this reason, the fill-in block F_i is saved in place of G_i and the subroutine essentially does not generate fill-in. The coefficient matrix is modified by the subroutine, but cannot be considered for solving other systems with different righthand-sides because the fill-in blocks are not saved and the permutation matrices not output.

5. COMPARISON WITH OTHER APPROACHES

In this section we analyze the reduction process in the BABDCR algorithm on the Wright example in Wright [1993]. Moreover, we compare our method with ABD packages applied to the doubled system (we are not aware of any available sequential code specifically designed to solve BABD systems).

The Wright example shows that the simple well-conditioned BABD matrix

$$\begin{pmatrix} I & & & & I \\ -C & I & & & \\ & -C & I & & \\ & & \ddots & \ddots & \\ & & & -C & I \end{pmatrix}, \tag{13}$$

may give rise to instability when factored by means of the LU factorization with row partial pivoting. In fact, suppose that the matrix (13) arises from the numerical solution of the well-conditioned BVP-ODE with nonseparated BCs

$$\begin{aligned} &y'(x) = Ay(x) + r(x), \quad A = \begin{pmatrix} -\frac{1}{6} & 1 \\ 1 & -\frac{1}{6} \end{pmatrix} \quad \text{in } x \in [0, 60], \\ &y(0) + y(60) = \eta, \end{aligned} \tag{14}$$

by means of multiple shooting. Then $C = \mathrm{e}^{hA}$, where h is the stepsize used to discretize the ODE problem. If h is such that the elements of C have modulus less than 1 (e.g., $h = 0.3$ and NBLOKS $= 200$ in Eq. (14)), then no permutation is ever performed and the right upper corner block implies fill-in in the last block column of the matrix U of the factorization, with elements $C, C^2, C^3, \ldots$ (see Wright [1993]). The same drawback appears when e^{hA} is approximated by means of the trapezoidal rule, thus obtaining $C = (I - hA/2)^{-1}(I + hA/2)$. Since one of the eigenvalues has modulus greater than 1, there is an exponential growth in the elements of the last block column, and the solution becomes incorrect, even if a moderate number of meshpoints is used.

For this reason, Garrett and Gladwell [2001] suggest solving BABD systems by means of an orthogonal factorization, thus obtaining a stable factorization, but with very large computational cost and memory requirement. In fact, QR is much more expensive than LU factorization and, in addition, since this algorithm is not specifically designed for BABD matrices, implies fill-in both on the last block column and on a further upper diagonal of the matrix R of the factorization. In such a case, a block scaling of the coefficient matrix (in order to obtain some blocks equal to the identity) before computing the QR factorization reduces the number of operations, but may be dangerous if performed on ill-conditioned blocks.

Table I. Operation Counts for the Solution of the BABD System (2)

	Factorization	Solution	Required Memory
BABDCR	$14/3 * \text{NRWBLK}^3$	$6 * \text{NRWBLK}^2$	$3 * \text{NRWBLK}^2$
COLROW	$46/3 * \text{NRWBLK}^3$	$16 * \text{NRWBLK}^2$	$8 * \text{NRWBLK}^2$
SOLVEBLOK	$76/3 * \text{NRWBLK}^3$	$20 * \text{NRWBLK}^2$	$12 * \text{NRWBLK}^2$

Each term must be multiplied by NBLOKS.

The BABDCR algorithm does not exhibit instability when applied to the Wright problem. In fact, the first step of reduction gives a matrix with the same structure as matrix (13) and C^2, instead of C ($P_i = I$ and $G_i = -C$ in Eq. (7)). Similarly, the successive steps give matrices $S^{(k)}$ with increasing powers of C (C^s, $s = 2^k$, after k steps), until the elements of the blocks become larger than 1 in modulus. At this point, since row pivoting is performed on the $2 * \text{NRWBLK}$ by NRWBLK block $\begin{pmatrix} I \\ C^s \end{pmatrix}$, the permutation allows control of the growth of elements of the new reduced matrices. For example, if $h = 0.3$, there is no permutation for the first step and the matrices $S^{(k)}$ and $R^{(k)}$ converge to

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

In Ascher et al. [1995] and Garrett and Gladwell [2001], a further possibility is considered to solve BABD systems by using the existing ABD solvers. Following this approach we introduce NBLOKS new unknowns y_i such that $x_0 = y_0 = \cdots = y_{\text{NBLOKS}}$ and obtain the new equations

$$y_0 = x_0, \quad y_{i+1} = y_i, \quad \text{for } i = 0, \ldots, \text{NBLOKS} - 1;$$

the boundary condition may be replaced by $B_a\, y_{\text{NBLOKS}} + B_b x_{\text{NBLOKS}} = f_0$. This means that, reassembling the unknowns and the equations appropriately, the linear system obtained has an ABD structure and may be treated by using any specific software for these problems. Garrett and Gladwell [2001] have tested SOLVEBLOK and the methods in Lam [1974] and Varah [1976]. These approaches do not show instability when applied to the Wright problem. However, the doubling of the size of each block required in these methods implies a larger computational cost than the BABDCR method. Table I summarizes the computational cost required by the BABDCR solver and packages COLROW and SOLVEBLOK applied to the rearranged ABD system.

ACKNOWLEDGMENTS

We thank the staff of the Department of Mathematics of the Southern Methodist University (Dallas, Texas, USA) who have provided technical support for completing this work during a semester of research spent by Giuseppe Romanazzi. The authors gratefully acknowledge, especially, Ian Gladwell for his comments.

REFERENCES

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

AMODIO, P. AND PAPRZYCKI, M. 1997. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput. 18*, 1, 56–68.

ASCHER, U. M., MATTHEIJ, R. M., AND RUSSELL, R. D. 1995. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Classics in Applied Mathematics 13. SIAM Press, Philadelphia, PA.

BRANKIN, R. AND GLADWELL, I. 1990. Codes for almost block diagonal systems. *Comput. Math. Appl. 19*, 7, 1–6.

DE BOOR, C. AND WEISS, R. 1980. Solveblok: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Softw. 6*, 80–87.

DIAZ, J., FAIRWEATHER, G., AND KEAST, P. 1983a. 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*, 3, 376–380.

DIAZ, J., FAIRWEATHER, G., AND KEAST, P. 1983b. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Softw. 9*, 3, 358–375.

FAIRWEATHER, G. AND GLADWELL, I. 2004. Algorithms for almost block diagonal linear systems. *SIAM Rev. 46*, 1, 49–58.

GARRETT, B. AND GLADWELL, I. 2001. Solving bordered almost block diagonal systems stably and efficiently. *J. Comput. Meth. Sci. Eng. 1*, 75–98.

HIGHAM, N. J. 1989. Algorithm 674: FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Softw. 15*, 2, 168.

LAM, D. 1974. Implemantation of the box scheme and model analysis of diffusion—convenction equations. Ph.D. thesis, University of Waterloo, Waterloo, Canada.

LENTINI, M. AND PEREYRA, V. 1977. An adaptive finite difference solver for nonlinear two-point boundary problems with mild boundary layers. *SIAM J. Numer. Anal. 14*, 91–111.

MAJAESS, F., KEAST, P., FAIRWEATHER, G., AND BENNETT, K. R. 1992a. Algorithm 704: ABDPACK and ABBPACK-FORTRAN programs for the solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. *ACM Trans. Math. Softw. 18*, 2, 205–210.

MAJAESS, F., KEAST, P., FAIRWEATHER, G., AND BENNETT, K. R. 1992b. The solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions. *ACM Trans. Math. Softw. 18*, 2, 193–204.

MUIR, P., PANCER, R., AND JACKSON, K. 2003. Pmirkdc: A parallel mono-implicit Runge–Kutta code with defect control for boundary value odes. *Parallel Comput. 29*, 6, 711–741.

VARAH, J. 1976. Alternate row and column elimination for solving certain linear systems. *SIAM J. Numer. Anal. 13*, 71–75.

WRIGHT, S. 1992. Stable parallel algorithms for two-point boundary value problems. *SIAM J. Sci. Stat. Comput. 13*, 3, 742–764.

WRIGHT, S. 1993. A collection of problems for which gaussian elimination with row partial pivoting is unstable. *SIAM J. Sci. Stat. Comp. 14*, 231–238.

WRIGHT, S. 1994. Stable parallel elimination for boundary value odes. *Numer. Math. 67*, 4, 521–535.

Received September 2004; revised November 2005; accepted December 2005