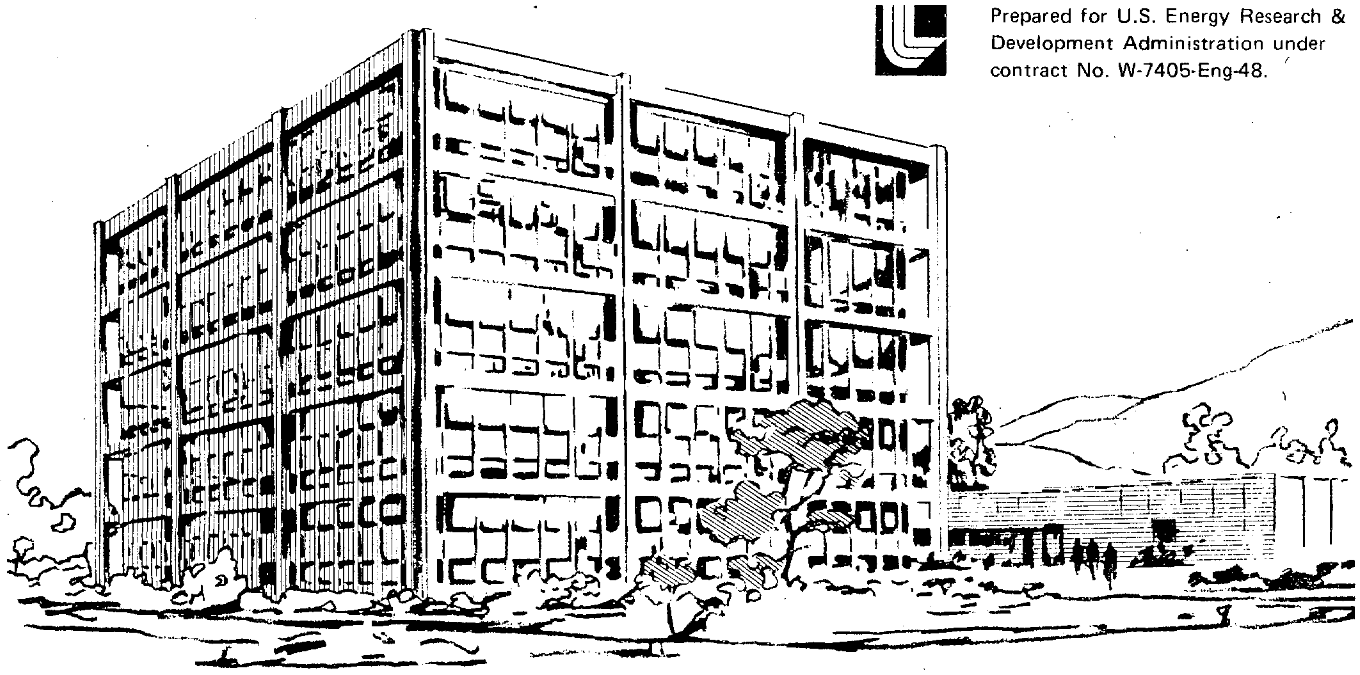
UCID-30150
COMPUTER
DOCUMENTATION

Lawrence Livermore Laboratory

SOLUTION OF BLOCK-TRIDIAGONAL SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

A. C. Hindmarsh

February 1977



Prepared for U.S. Energy Research & Development Administration under contract No. W-7405-Eng-48.

C O N T E N T S

SOLUTION OF BLOCK-TRIDIAGONAL SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

A.C. Hindmarsh

December, 1976

ABSTRACT

The solution of linear systems with a block-tridiagonal structure is a very common requirement in many applications. This report describes two Fortran packages for solving such systems. The first is for the case when all the relevant blocks in the coefficient matrix can be stored at once. The second is for the case when they cannot, and instead the equations are generated, one block-row at a time, by a user-supplied subroutine. The blocks are assumed to be full, and partial pivoting is done within block-rows only.

I N T R O D U C T I O N

This report, and the routines given here, are concerned with the solution of systems of linear algebraic equations,

$$T\, x = y \quad , \tag{1}$$

in which the coefficient matrix T is square and block-tridiagonal. By this we mean that T has the block structure

$$T = \begin{bmatrix} A_1 & B_1 & C_1 & & & \bigcirc \\ C_2 & A_2 & B_2 & & & \\ & C_3 & A_3 & B_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & C_{N-1} & A_{N-1} & B_{N-1} \\ \bigcirc & & & B_N & C_N & A_N \end{bmatrix} . \tag{2}$$

In terms of the individual blocks of equations, (1) and (2) can be written as

$$\left.\begin{aligned} &A_1\, x_1 + B_1\, x_2 + C_1\, x_3 = y_1 \\ &C_k\, x_{k-1} + A_k\, x_k + B_k\, x_{k+1} = y_k \qquad (2 \le k \le N-1) \\ &B_N\, x_{N-2} + C_N\, x_{N-1} + A_N\, x_N = y_N \end{aligned}\right\} . \tag{3}$$

Here x_i and y_k denote vectors of length M in the i^{th} block position of x and y, respectively. This system is a slight generalization of the usual meaning of a block-tridiagonal system, in that blocks are allowed in the (1,3) and (N,N-2) block positions, namely C_1 and B_N, respectively. It is assumed that $N \geq 4$, since otherwise there is no structure assumed for the matrix T which can be utilized to any advantage.

Systems of this form are found frequently in various applications, chiefly in the numerical solution of systems of partial differential equations. The presence of C_1 and B_N is due to certain types of boundary conditions for such problems. The particular application which motivated this work was to the systems that arise in ADI (alternating direction implicit) methods for two-dimensional problems, as exemplified by [1].

All of the blocks A_i, B_i, C_i in (2) are treated here as full $M \times M$ matrices. If N is the number of blocks in each direction in this structure, then the order of the matrix T is MN, and the total number of nonzero elements allowed in T is $3M^2N$.

The two packages presented here both solve the system (1) by performing an appropriate decomposition of the matrix T and performing back-substitutions to solve for x. The decomposition is structured block-LU decomposition, described in detail below, in the subsection called Method in Part I. The algorithm involves the solution of many $M \times M$ linear systems, and for this the dense linear system routines DEC and SOL, given in [2], are used.

The need for two distinct packages arises from the storage and efficiency considerations associated with various contexts in which block-tridiagonal systems are solved. In the first package, called BT, it is assumed that the problem input, T and y, can all be stored in core at once, involving $3M^2N + MN$ elements. The LU factorization is then saved, and this allows the solutions of (1) and any subsequent systems, with the same matrix T but different vectors y, to be accomplished very cheaply. However, for contexts in which that much storage is not available, there is a minimal storage version of the package, called BTMS. It requires only $M^2N + MN + 3M^2$ words of storage, and calls for the blocks of elements as they are needed. It is just as efficient as the first package for the solution of a single system, but it offers no cost savings at all for subsequent systems.

Both of the packages given here are written in ANSI Standard Fortran, in order to allow for maximum portability. They use single precision exclusively. The auxiliary routines DEC and SOL, however, are available in a vectorized version, an assembly language (Compass) version, and a version for the CDC-STAR, as well as a standard Fortran version. On the CDC-7600, considerable improvements in efficiency are possible by substituting the faster versions of DEC/SOL in the packages given here.

P A R T I.

FULL STORAGE VERSION (BT)

Availability

The package referred to here as BT consists of the two subroutines DECBT and SOLBT, which are listed in Appendix I for reference, and the auxiliary routines DEC and SOL [2]. At LLL, the Fortran source code for the BT package is available from the NMG Mathematical Software Library [3], by way of the access routine MSLAR, using the teletype command

```
NMG MSLAR READ DECBT END (return)
```

See [2] for instructions on obtaining the non-standard versions of DEC/SOL, if desired. The routines contain OPTIMIZE cards, which must be removed for use under systems other than CHAT or ORDER.

A demonstration program is also available. It solves an example problem, as described below. The source code for this program, together with the BT package, is obtainable with the command

```
NMG MSLAR READ DEMO DECBT END (return)
```

Usage

The BT package is used by making calls to DECBT and SOLBT. Subroutine DECBT is to be called once only for a given matrix T, and performs an appropriate decomposition of T without reference to the vectors x or y. Subroutine SOLBT is to be called once for each right-hand side vector y in the system Tx = y, where DECBT has been previously called for T. The calling sequences for the two routines are as follows:

```
CALL DECBT (M, N, A, B, C, IP, IER)
[Test IER]
CALL SOLBT (M, N, A, B, C, Y, IP)
```

The definitions of the arguments are as follows:

M = the order (size) of each block in T.

N = the number of blocks in each direction along the matrix T, with $N \geq 4$. The total order of T is M*N.

A = an $M \times M \times N$ array containing the diagonal blocks $A_1, \ldots, A_N$, in the notation of (2), on input to DECBT. A(I,J,K) must be set to the (I,J) element of A_K. The first and second dimension of the A array are assumed to be exactly M, so that the elements are stored in packed form, in the usual (columnwise) order.

B = an $M \times M \times N$ array containing the blocks $B_1, \ldots, B_N$, on input to DECBT, stored in the same manner as in A.

C = an $M \times M \times N$ array containing the blocks $C_1, \ldots, C_N$, on input to DECBT, stored in the same manner as in A.

IP = an integer array of length M*N for working storage (used for pivot information).

IER = an output error indicator from DECBT.

IER = 0 if no errors occurred;

= -1 if the input value of M or N was illegal; and

= K if a singular diagonal block was found in the K^{th} stage of the decomposition.

SOLBT should not be called if IER $\neq$ 0.

Y = the right-hand side vector y, of length M*N, on input to SOLBT, and the solution vector x on output.

The arrays A, B, C, and IP, on output from DECBT, contain the block-LU decomposition of T, and so must not be disturbed between a call to DECBT and any subsequent calls to SOLBT for a given matrix T. For the same reason, if the user wishes to save the matrix T, the arrays A, B, and C must be copied to separate locations before the call to DECBT.

The SOLBT routine must not be called if IER $\neq 0$ on return from DECBT, as this will produce nonsense. If IER = 0, as many calls to SOLBT can be made as there are distinct vectors y for the same matrix T.

Example

To illustrate the use of the BT package, we consider a small example problem with M = 3 and N = 10. We take

$$A_k = \begin{bmatrix} -8 & 1 & 0 \\ 1 & -8 & 1 \\ 0 & 1 & -8 \end{bmatrix} \quad , \quad B_k = C_k = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

for k = 1, 2, ..., N. For the purposes of the illustration, an exact solution vector x^e is taken to be

$$x^e = (1, 2, 3, \ldots, 30)^T \quad ,$$

and the right-hand side vector y is Tx^e. The following demonstration program generates the input to the package, calls the package, and prints the answers. The vector y, which was computed independently, is data-loaded in this program. The output for this program (not shown here) shows a maximum error in the components of x of less than 10^{-12} under either CHAT or PUTT.

```
C
C DEMONSTRATION PROGRAM FOR BT PACKAGE.
C
C THIS PROGRAM GENERATES A 30 BY 30 BLOCK-TRIDIAGONAL LINEAR SYSTEM,
C COMPUTES THE SOLUTION WITH DECBT/SOLBT, AND PRINTS ANSWERS ON
C LOGICAL UNIT 3.  AN EXACT SOLUTION VECTOR IS PRESET, AND THE
C CORRESPONDING RIGHT-HAND SIDE VECTOR IS DATA-LOADED.
C
      DIMENSION A(3,3,10),B(3,3,10),C(3,3,10),XE(30),Y(30),IP(30)
      DATA Y /11.,1.,-13.,-13.,-20.,-37.,-28.,-32.,-52.,-43.,
     1     -44.,-67.,-58.,-56.,-82.,-73.,-68.,-97.,-88.,-80.,
     2     -112.,-103.,-92.,-127.,-118.,-104.,-142.,-142.,-125.,-166./
      M = 3
      N = 10
C
C GENERATE MATRIX.
      DO 30 K = 1,N
        DO 20 J = 1,M
          DO 10 I = 1,M
            A(I,J,K) = 1.
            B(I,J,K) = 1.
 10         C(I,J,K) = 1.
          A(J,J,K) = -8.
          B(J,J,K) = -1.
 20       C(J,J,K) = -1.
        A(1,3,K) = 0.
        A(3,1,K) = 0.
 30     CONTINUE
C
C GENERATE EXACT SOLUTION VECTOR XE.
      MN = M*N
      DO 40 I = 1,MN
 40     XE(I) = FLOAT(I)
C
C CALL DECBT AND SOLBT AND WRITE ANSWERS.
      WRITE(3,100)M,N
 100  FORMAT(37H DEMONSTRATION PROGRAM FOR BT PACKAGE//
     1   4H M =,I4,6H   N =,I4//)
      CALL DECBT (M, N, A, B, C, IP, IER)
      IF (IER .EQ. 0) GO TO 120
C
      WRITE(3,110)IER
 110  FORMAT(//22H DECBT FAILED..  IER =,I5//)
      GO TO 200
C
 120  CALL SOLBT (M, N, A, B, C, Y, IP)
      WRITE(3,130)(Y(I),I=1,MN)
 130  FORMAT(22H COMPUTED SOLUTION Y =/(9E12.3))
      ERM = 0.
      DO 140 I = 1,MN
        ERI = ABS(Y(I) - XE(I))
 140    ERM = AMAX1(ERM,ERI)
      WRITE(3,150)ERM
 150  FORMAT(//18H MAX. ERROR IN Y =,E20.12)
 200  CALL EXIT
      END
```

<u>Method</u>

The basic method used to solve the system (1) and (2) is Gaussian elimination, formulated as a block-LU decomposition of T followed by back-substitutions. For the decomposition, we write T in the factored form

$$T = LU = \begin{bmatrix} D_1 & & & & \bigcirc \\ C_2 & D_2 & & & \\ & \ddots & \ddots & & \\ & & C_{N-1} & D_{N-1} & \\ \bigcirc & B_N & & C'_N & D_N \end{bmatrix} \begin{bmatrix} I & E_1 & C'_1 & & \bigcirc \\ & I & E_2 & & \\ & & \ddots & \ddots & \\ & & & I & E_{N-1} \\ \bigcirc & & & & I \end{bmatrix} , \qquad (4)$$

where I represents the $M \times M$ identity matrix. The factor L is block-lower triangular, and the factor U is unit-upper triangular (because of the unit diagonal). While not guaranteed to exist in all cases, this decomposition exists in most (if not all) cases of practical interest. The various blocks in (4) can be computed, in succession, by the relations

$$\left.\begin{array}{l} D_1 = A_1, \quad E_1 = D_1^{-1}B_1, \quad C'_1 = D_1^{-1}C_1, \\ D_2 = A_2 - C_2E_1, \quad E_2 = D_2^{-1}(B_2 - C_2C'_1) , \\ D_k = A_k - C_kE_{k-1}, \quad E_k = D_k^{-1}B_k \quad (k = 3, 4, \ldots, N-1), \\ C'_N = C_N - B_NE_{N-2}, \quad D_N = A_N - C'_NE_{N-1} \; . \end{array}\right\} \qquad (5)$$

These relations can be derived simply by comparing (4) with (2), block by block. The sequence of operations in (5) is performed by DECBT, and the results are written into the arrays A, B, and C, overwriting T. Wherever (5) calls for a matrix D_k^{-1}, the LU decomposition of D_k is computed by DEC (in the space originally occupied by A_k), and multiplications by D_k^{-1} are accomplished by calls to SOL. Inverse matrices are <u>not</u> computed explicitly here. The other nonzero blocks in

(4) (excluding I) are written, as computed, into the locations occupied by the corresponding blocks (in B and C) in (2).

The computation of the LU decomposition of each D_k involves partial pivoting, meaning that rows are interchanged in a manner designed to control the growth of roundoff error. The interchanges performed are recorded in the IP array. However, these pivoting operations are done only within each block of rows of T separately. For this reason, it is conceivable that the overall algorithm (5) may suffer from numerical instability (i.e., harmful growth of roundoff errors), if T is particularly unsuitable. That is, even though T may be nonsingular, and the system (1) may be perfectly well-posed, it may happen that in the algorithm (5), some D_k is a singular or nearly singular matrix, and the subsequent computations may either be impossible or produce highly inaccurate answers. If this happens in such a way that D_k is actually found to be singular by DEC, the process is halted and DECBT returns IER = k. However, a near-singularity in D_k may go undetected. This warning is not necessary, however, if T has appropriate properties (e.g., if T is diagonally dominant*). In practical applications, these properties tend to hold in the matrices of interest. Thus, numerical instability is not likely to arise in practice.

Once the decomposition (4) is accomplished, the solution of the system (1) is rather easily obtained. As in (3), denote the blocks in x and y by x_i and y_i, respectively. We then proceed to solve Lz = y and Ux = z in succession, by the operations

*A matrix $T = (t_{ij})$ is called diagonally dominant if $|t_{ii}| \geq \Sigma_{j \neq i} |t_{ij}|$ with strict inequality for at least one value of i.

$$\left.\begin{aligned} z_1 &= D_1^{-1}\, y_1 \\ z_k &= D_k^{-1}\,(y_k - C_k\, z_{k-1}) \qquad (k = 2, 3, \ldots, N-1) \\ z_N &= D_N^{-1}\,(y_N - C_N'\, z_{N-1} - B_N\, z_{N-2}) \end{aligned}\right\} \tag{6}$$

$$\left.\begin{aligned} x_N &= z_N \\ x_k &= z_k - E_k\, x_{k+1} \qquad (k = N-1, N-2, \ldots, 2) \\ x_1 &= z_1 - E_1\, x_2 - C_1'\, x_3 \end{aligned}\right\} . \tag{7}$$

These operations are performed by SOLBT, which writes z in place of y and then x in place of z. The multiplication of a vector by D_k^{-1} in (6) is done by SOL with the existing LU decomposition of D_k.

P A R T I I .

MINIMAL STORAGE VERSION (BTMS)

Availability

The BTMS package consists of subroutine BTMS, which is listed in Appendix II for reference, and the auxiliary routines DEC and SOL [2]. At LLL, the Fortran source code for the package is available from the NMG Mathematical Software Library [3], by way of the access routine MSLAR, using the teletype command

```
NMG MSLAR READ BTMS END (return)
```

See [2] for instructions on obtaining the non-standard versions of DEC/SOL, if desired. The routines contain OPTIMIZE cards, which must be removed for use under systems other than CHAT or ORDER.

A demonstration program is also available. It solves an example problem, as described below. The source code for this program, together with the BTMS package, is obtainable with the command

```
NMG MSLAR READ DEMO BTMS END (return)
```

Usage

The BTMS package is used by making calls to subroutine BTMS and also supplying a subroutine called BLOX to communicate the elements of the matrix T and the vector y. The calling sequence for BTMS is as follows:

```
CALL BTMS (M, N, E, A, B, C, IP, X, IER)
```

The definitions of the arguments are as follows.

M = the order (size) of each block in T.

N	= the number of blocks in each direction along the matrix T, with $N \geq 4$. The total order of T is M*N.
E	= a working storage array of length M*M*N.
A,B,C	= working storage arrays, each of length M*M.
IP	= an integer array of length M for working storage (used for pivot information). To save on storage, the IP array may occupy the same space as part of the B array.
X	= an array containing, on output, the solution vector x, of length M*N.
IER	= an output error indicator.

IER = 0 if no errors occurred;

= -1 if the input value of M or N was illegal; and

= K if a singular diagonal block was found in the Kth stage of the decomposition.

If IER $\neq$ 0, a solution vector was <u>not</u> computed.

The user must provide a subroutine, BLOX, to supply T and y to BTMS. The argument sequence of BLOX is as follows:

```
SUBROUTINE BLOX (M, K, A, B, C, YK)
```

The arguments M and K are input, M being as before, and K being an index, $1 \leq K \leq N$. Subroutine BLOX is then to load into A, B, and C the $M \times M$ matrix blocks of the T matrix in the Kth block-row, namely A_K, B_K, and C_K in the notation of (2) or (3), and load into the vector YK the block (of length M) of y in the Kth block-row, namely y_K in (3). A, B, and C must be given a first dimension of M.

If it is more convenient, the vector y may be loaded into the X array prior to the call to BTMS, rather than loaded by BLOX.

The required storage for data arrays in BTMS is $M^2N + MN + 3M^2$. This is to be compared with the requirement of $3M^2N + 2MN$ for BT. For large N, the difference in storage costs may be considerable, since the dominant term in the cost is one-third as large for BTMS.

If there is only one linear system (1) that must be solved (i.e., one vector y) for a given value of the matrix T, the computational cost of solving it with BTMS is virtually the same as with BT. Hence, in that case, the storage advantage may weigh heavily in favor of BTMS. However, if there is more than one such system for a given T, the computational cost of solving them with BTMS can be much greater than with BT. This is because each such solution has the same cost with BTMS, while with BT solutions after the first one are much cheaper than the first one. Thus, there is a tradeoff of run-time vs. storage which must be carefully considered.

Example

We illustrate the use of BTMS with the same small example problem ($M = 3$, $N = 10$) given in Part I above. The following demonstration program calls BTMS to solve the problem, and contains Subroutine BLOX to provide the input matrix and vector. The output (not shown here) shows a maximum error in the components of x of less than 10^{-12}, under either CHAT or PUTT.

```
C
C DEMONSTRATION PROGRAM FOR BTMS PACKAGE.
C
C THIS PROGRAM GENERATES A 30 BY 30 BLOCK-TRIDIAGONAL LINEAR SYSTEM,
C COMPUTES THE SOLUTION WITH BTMS, AND PRINTS ANSWERS ON
C LOGICAL UNIT 3.  THE EXACT SOLUTION VECTOR IS X(I) = I,
C AND THE CORRESPONDING RIGHT-HAND SIDE VECTOR IS DATA-LOADED.
C
      DIMENSION E(90), A(9), B(9), C(9), X(30), IP(3)
      COMMON /RHSVEC/ RHS(30)
      DATA RHS /11.,1.,-13.,-13.,-20.,-37.,-28.,-32.,-52.,-43.,
     1      -44.,-67.,-58.,-56.,-82.,-73.,-68.,-97.,-88.,-80.,
     2      -112.,-103.,-92.,-127.,-118.,-104.,-142.,-142.,-125.,-166./
      M = 3
      N = 10
      MN = M*N
C
C
      WRITE(3,100)M,N
 100  FORMAT(39H DEMONSTRATION PROGRAM FOR BTMS PACKAGE//
     1   4H M =,I4,6H    N =,I4//)
C
      CALL BTMS (M, N, E, A, B, C, IP, X, IER)
C
      IF (IER .EQ. 0) GO TO 120
C
      WRITE(3,110)IER
 110  FORMAT(//22H DECBT FAILED..  IER =,I5//)
      GO TO 200
C
 120  WRITE(3,130)(X(I),I=1,MN)
 130  FORMAT(22H COMPUTED SOLUTION X =/(9E12.3))
      ERM = 0.
      DO 140 I = 1,MN
         ERI = ABS(X(I) - FLOAT(I))
 140     ERM = AMAX1(ERM,ERI)
      WRITE(3,150)ERM
 150  FORMAT(//18H MAX. ERROR IN X =,E20.12)
 200  CALL EXIT
      END
```

```
      SUBROUTINE BLOX (M, K, A, B, C, YK)
      DIMENSION A(M,M), B(M,M), C(M,M), YK(M)
      COMMON /RHSVEC/ RHS(30)
      DO 20 J = 1,M
        DO 10 I = 1,M
          A(I,J) = 1.
          B(I,J) = 1.
 10       C(I,J) = 1.
        A(J,J) = -8.
        B(J,J) = -1.
        C(J,J) = -1.
 20     CONTINUE
      A(1,3) = 0.
      A(3,1) = 0.
      DO 30 I = 1,M
        J = (K-1)*M + I
 30     YK(I) = RHS(J)
      RETURN
      END
```

Method

The method used by BTMS is exactly the same as in BT —block-LU decomposition of T followed by back substitution to solve for x. The difference is in the order of the operations performed and in the interface with the user. The decomposition operations (5) and the forward substitution operations (6) are merged together in such a way that a minimum of working space is needed. The only blocks of the matrix that are saved in this sequence of operations are the blocks in the upper triangular factor U, namely E_1, ..., E_{N-1}, and C_1'. These are stored in the array E, with C_1' occupying the Nth block. Then the backward substitution operations (7) are performed to get x. As in BT, multiplications by D_k^{-1} are accomplished by the LU method (with DEC/SOL), but the LU decompositions of the D_k are saved only as long as they are needed in the algorithm.

The working storage in BTMS is arranged in such a way that the B array is used only in the processing of the last block-row in (3). As a result, if storage is critical and $B_N = 0$, it is easy to modify the package so that the B array is not required at all. However, if the IP array overlaps B in the user's call to the unmodified package, then a separate IP array must be supplied to the modified version. The savings in storage is then $M^2 - M$.

TESTING

Both the BT and BTMS packages were subjected to several series of tests. Test problems were generated by choosing random elements for T and x in the interval (-1,1), except for the diagonal elements of T, which were made large enough to insure diagonal dominance. Then right-hand side vectors y were set to Tx, and the solution vector computed by the package was compared to the pre-set x. Runs were made with $1 \leq M \leq 9$ and $4 \leq N \leq 50$. In all cases, the errors were at the level of machine roundoff (less than 10^{-13}). Additional tests were made to check the tests for illegal input and for a singular matrix, and to validate the packages on PUTT as well as CHAT. The use of the Compass version of DEC/SOL reduced run times by factors of, typically, about 1.5.

As a sample of the run times, a test problem with M = 6 and N = 50 required about 68 millisec for either of the two packages, with the Fortran version of DEC/SOL, and about 45 millisec with the Compass version. With BTMS and the Fortran DEC/SOL, the run time was 96 ms with the OPTIMIZE card removed from BTMS. The run times for DECBT/SOLBT are about 3% lower than for BTMS (all else being equal), because of the overhead of the calls to BLOX in the latter.

Details of these tests are on file with the Numerical Mathematics Group's Test Files.

REFERENCES

[1] I. Lindemuth and J. Killeen, Alternating Direction Implicit Techniques for Two Dimensional Magnetohydrodynamic Calculations, J. Comp. Phys., 13 (1973), pp. 181-208.

[2] A.C. Hindmarsh, L.J. Sloan, K.W. Fong, and G.H. Rodrigue, DEC/SOL: Solution of Dense Systems of Linear Algebraic Equations, LLL Report UCID-30137 (1976).

[3] F. N. Fritsch and A. W. Hall, Numerical Mathematics Group Mathematical Software Library Catalog, LLL Report UCID-30136 (1976).

A P P E N D I X   I

LISTING OF DECBT / SOLBT

```
      SUBROUTINE DECBT (M, N, A, B, C, IP, IER)
      DIMENSION A(M,M,N), B(M,M,N), C(M,M,N), IP(M,N)
C-----------------------------------------------------------------------
C  THE FOLLOWING CARD IS FOR OPTIMIZED COMPILATION UNDER CHAT.
      OPTIMIZE
C-----------------------------------------------------------------------
C BLOCK-TRIDIAGONAL MATRIX DECOMPOSITION ROUTINE.
C WRITTEN BY A. C. HINDMARSH.
C THE INPUT MATRIX CONTAINS THREE BLOCKS OF ELEMENTS IN EACH BLOCK-ROW,
C INCLUDING BLOCKS IN THE (1,3) AND (N,N-2) BLOCK POSITIONS.
C DECBT USES BLOCK GAUSS ELIMINATION AND SUBROUTINES DEC AND SOL
C FOR SOLUTION OF BLOCKS.  PARTIAL PIVOTING IS DONE WITHIN
C BLOCK-ROWS ONLY.
C INPUT..
C      M = ORDER OF EACH BLOCK.
C      N = NUMBER OF BLOCKS IN EACH DIRECTION OF THE MATRIX.
C          N MUST BE 4 OR MORE.  THE COMPLETE MATRIX HAS ORDER M*N.
C      A = M BY M BY N ARRAY CONTAINING DIAGONAL BLOCKS.
C          A(I,J,K) CONTAINS THE (I,J) ELEMENT OF THE K-TH BLOCK.
C      B = M BY M BY N ARRAY CONTAINING THE SUPER-DIAGONAL BLOCKS
C          (IN B(,,K) FOR K = 1,...,N-1) AND THE BLOCK IN THE (N,N-2)
C          BLOCK POSITION (IN B(,,N)).
C      C = M BY M BY N ARRAY CONTAINING THE SUBDIAGONAL BLOCKS
C          (IN C(,,K) FOR K = 2,3,...,N) AND THE BLOCK IN THE
C          (1,3) BLOCK POSITION (IN C(,,1)).
C     IP = INTEGER ARRAY OF LENGTH M*N FOR WORKING STORAGE.
C OUTPUT..
C A,B,C = M BY M BY N ARRAYS CONTAINING THE BLOCK LU DECOMPOSITION
C          OF THE INPUT MATRIX.
C     IP = M BY N ARRAY OF PIVOT INFORMATION.  IP(,K) CONTAINS
C          INFORMATION FOR THE K-TH DIGONAL BLOCK.
C    IER = 0  IF NO TROUBLE OCCURRED, OR
C        = -1 IF THE INPUT VALUE OF M OR N WAS ILLEGAL, OR
C        = K  IF A SINGULAR MATRIX WAS FOUND IN THE K-TH DIAGONAL BLOCK.
C USE SOLBT TO SOLVE THE ASSOCIATED LINEAR SYSTEM.
C DECBT CALLS SUBROUTINES  DEC(M,MO,A,IP,IER)  AND  SOL(M,MO,A,Y,IP)
C FOR SOLUTION OF M BY M LINEAR SYSTEMS.
C-----------------------------------------------------------------------
      IF (M .LT. 1 .OR. N .LT. 4) GO TO 210
      NM1 = N - 1
      NM2 = N - 2
C PROCESS THE FIRST BLOCK-ROW. -----------------------------------------
      CALL DEC (M, M, A, IP, IER)
      K = 1
      IF (IER .NE. 0) GO TO 200
      DO 10 J = 1,M
        CALL SOL (M, M, A, B(1,J,1), IP)
        CALL SOL (M, M, A, C(1,J,1), IP)
 10     CONTINUE
C ADJUST B(,,2). -------------------------------------------------------
      DO 40 J = 1,M
        DO 30 I = 1,M
          DP = 0.
          DO 20 L = 1,M
 20         DP = DP + C(I,L,2)*C(L,J,1)
 30       B(I,J,2) = B(I,J,2) - DP
 40     CONTINUE
```

```
C MAIN LOOP.  PROCESS BLOCK-ROWS 2 TO N-1. -----------------------------
      DO 100 K = 2,NM1
        KM1 = K - 1
        DO 70 J = 1,M
          DO 60 I = 1,M
            DP = 0.
            DO 50 L = 1,M
 50           DP = DP + C(I,L,K)*B(L,J,KM1)
 60         A(I,J,K) = A(I,J,K) - DP
 70       CONTINUE
        CALL DEC (M, M, A(1,1,K), IP(1,K), IER)
        IF (IER .NE. 0) GO TO 200
        DO 80 J = 1,M
 80       CALL SOL (M, M, A(1,1,K), B(1,J,K), IP(1,K))
 100    CONTINUE
C PROCESS LAST BLOCK-ROW AND RETURN. ------------------------------------
      DO 130 J = 1,M
        DO 120 I = 1,M
          DP = 0.
          DO 110 L = 1,M
 110        DP = DP + B(I,L,N)*B(L,J,NM2)
 120      C(I,J,N) = C(I,J,N) - DP
 130    CONTINUE
      DO 160 J = 1,M
        DO 150 I = 1,M
          DP = 0.
          DO 140 L = 1,M
 140        DP = DP + C(I,L,N)*B(L,J,NM1)
 150      A(I,J,N) = A(I,J,N) - DP
 160    CONTINUE
      CALL DEC (M, M, A(1,1,N), IP(1,N), IER)
      K = N
      IF (IER .NE. 0) GO TO 200
      RETURN
C ERROR RETURNS. --------------------------------------------------------
 200  IER = K
      RETURN
 210  IER = -1
      RETURN
C----------------------- END OF SUBROUTINE DECBT --------------------
      END
      SUBROUTINE SOLBT (M, N, A, B, C, Y, IP)
C-----------------------------------------------------------------------
C  THE FOLLOWING CARD IS FOR OPTIMIZED COMPILATION UNDER CHAT.
      OPTIMIZE
C-----------------------------------------------------------------------
C SOLUTION OF BLOCK-TRIDIAGONAL LINEAR SYSTEM.
C COEFFICIENT MATRIX MUST HAVE BEEN PREVIOUSLY PROCESSED BY DECBT.
C INPUT..
C      M = ORDER OF EACH BLOCK.
C      N = NUMBER OF BLOCKS IN EACH DIRECTION OF MATRIX.
C A,B,C = M BY M BY N ARRAYS CONTAINING BLOCK LU DECOMPOSITION
C         OF COEFFICIENT MATRIX FROM DECBT.
C     IP = M BY N INTEGER ARRAY OF PIVOT INFORMATION FROM DECBT.
C      Y = ARRAY OF LENGTH M*N CONTAING THE RIGHT-HAND SIDE VECTOR
C          (TREATED AS AN M BY N ARRAY HERE).
C OUTPUT..
C      Y = SOLUTION VECTOR, OF LENGTH M*N.
C SOLBT MAKES CALLS TO SUBROUTINE SOL(M,MO,A,Y,IP)
C FOR SOLUTION OF M BY M LINEAR SYSTEMS.
C-----------------------------------------------------------------------
      DIMENSION A(M,M,N), B(M,M,N), C(M,M,N), Y(M,N), IP(M,N)
C
      NM1 = N - 1
      NM2 = N - 2
C FORWARD SOLUTION SWEEP. ----------------------------------------------
      CALL SOL (M, M, A, Y, IP)
      DO 30 K = 2,NM1
        KM1 = K - 1
        DO 20 I = 1,M
          DP = 0.
          DO 10 J = 1,M
 10         DP = DP + C(I,J,K)*Y(J,KM1)
 20       Y(I,K) = Y(I,K) - DP
        CALL SOL (M, M, A(1,1,K), Y(1,K), IP(1,K))
 30     CONTINUE
      DO 50 I = 1,M
        DP = 0.
        DO 40 J = 1,M
 40       DP = DP + C(I,J,N)*Y(J,NM1) + B(I,J,N)*Y(J,NM2)
 50     Y(I,N) = Y(I,N) - DP
      CALL SOL (M, M, A(1,1,N), Y(1,N), IP(1,N))
C BACKWARD SOLUTION SWEEP. ---------------------------------------------
      DO 80 KB = 1,NM1
        K = N - KB
        KP1 = K + 1
        DO 70 I = 1,M
          DP = 0.
          DO 60 J = 1,M
 60         DP = DP + B(I,J,K)*Y(J,KP1)
 70       Y(I,K) = Y(I,K) - DP
 80     CONTINUE
      DO 100 I = 1,M
        DP = 0.
        DO 90 J = 1,M
 90       DP = DP + C(I,J,1)*Y(J,3)
 100    Y(I,1) = Y(I,1) - DP
      RETURN
C----------------------- END OF SUBROUTINE SOLBT -----------------------
      END
```

A P P E N D I X   I I

LISTING OF BTMS

```
      SUBROUTINE BTMS (M, N, E, A, B, C, IP, X, IER)
      DIMENSION E(M,M,N), A(M,M), B(M,M), C(M,M), IP(M), X(M,N)
C-----------------------------------------------------------------------
C  THE FOLLOWING CARD IS FOR OPTIMIZED COMPILATION UNDER CHAT.
      OPTIMIZE
C-----------------------------------------------------------------------
C BLOCK-TRIDIAGONAL LINEAR SYSTEM SOLVER, MINIMAL STORAGE VERSION.
C WRITTEN BY A. C. HINDMARSH.
C THE COEFFICIENT MATRIX CONTAINS THREE BLOCKS OF ELEMENTS IN EACH
C BLOCK-ROW, INCLUDING BLOCKS IN THE (1,3) AND (N,N-2) BLOCK POSITIONS.
C THE BLOCKS OF THE MATRIX AND THE CORRESPONDING BLOCKS OF THE
C RIGHT-HAND SIDE VECTOR ARE OBTAINED, ONE BLOCK-ROW AT A TIME,
C FROM THE USER-SUPPLIED ROUTINE BLOX.
C BTMS USES BLOCK GAUSS ELIMINATION AND SUBROUTINES DEC AND SOL
C FOR SOLUTION OF BLOCKS.  PARTIAL PIVOTING IS DONE WITHIN
C BLOCK-ROWS ONLY.
C
C INPUT..
C      M = ORDER OF EACH BLOCK.
C      N = NUMBER OF BLOCKS IN EACH DIRECTION OF THE MATRIX.
C          N MUST BE 4 OR MORE.  THE COMPLETE MATRIX HAS ORDER M*N.
C      E = WORKING STORAGE ARRAY OF LENGTH M*M*N.
C A,B,C = WORKING STORAGE ARRAYS, EACH OF LENGTH M*M.
C     IP = INTEGER ARRAY OF LENGTH M FOR WORKING STORAGE.
C          THE IP ARRAY CAN OVERLAP THE B ARRAY.
C      X = WORKING STORAGE ARRAY OF LENGTH M*N.
C OUTPUT..
C      X = SOLUTION VECTOR, OF LENGTH M*N.
C    IER = 0  IF NO TROUBLE OCCURRED, OR
C        = -1 IF THE INPUT VALUE OF M OR N WAS ILLEGAL, OR
C        = K  IF A SINGULAR MATRIX WAS FOUND IN THE K-TH DIAGONAL BLOCK.
C
C THE USER MUST SUPPLY THE FOLLOWING..
C                SUBROUTINE BLOX (M, K, A, B, C, YK)
C THIS ROUTINE IS TO LOAD INTO A, B, AND C THE M BY M BLOCKS IN THE
C K-TH BLOCK-ROW OF THE COEFFICIENT MATRIX, AND INTO YK THE VECTOR
C OF LENGTH M IN THE K-TH BLOCK OF THE RIGHT-HAND SIDE VECTOR.
C IN THE COEFFICIENT MATRIX, THE BLOCK A IS ALWAYS THE ONE ON THE MAIN
C DIAGONAL.  THE K-TH BLOCK-ROW READS  C,A,B  FOR K = 2,...,N-1.
C FOR K = 1 IT READS  A,B,C  AND FOR K = N IT READS  B,C,A.
C
C BTMS CALLS SUBROUTINES  DEC(M,MO,A,IP,IER)  AND  SOL(M,MO,A,X,IP)
C FOR SOLUTION OF M BY M LINEAR SYSTEMS.
C-----------------------------------------------------------------------
      IF (M .LT. 1 .OR. N .LT. 4) GO TO 310
      NM1 = N - 1
      NM2 = N - 2
C READ AND PROCESS THE FIRST BLOCK-ROW. ---------------------------------
      K = 1
      CALL BLOX (M, K, A, E, E(1,1,N), X)
      CALL DEC (M, M, A, IP, IER)
      IF (IER .NE. 0) GO TO 300
      DO 10 J = 1,M
        CALL SOL (M, M, A, E(1,J,1), IP)
        CALL SOL (M, M, A, E(1,J,N), IP)
 10     CONTINUE
      CALL SOL (M, M, A, X, IP)
C READ AND PROCESS THE SECOND BLOCK ROW. --------------------------------
      K = 2
      CALL BLOX (M, K, A, E(1,1,2), C, X(1,2))
      DO 40 J = 1,M
        DO 30 I = 1,M
          DP = 0.
          DQ = 0.
          DO 20 L = 1,M
            DP = DP + C(I,L)*E(L,J,1)
 20         DQ = DQ + C(I,L)*E(L,J,N)
          A(I,J) = A(I,J) - DP
 30       E(I,J,2) = E(I,J,2) - DQ
 40     CONTINUE
      DO 60 I = 1,M
        DP = 0.
        DO 50 L = 1,M
 50       DP = DP + C(I,L)*X(L,1)
 60     X(I,2) = X(I,2) - DP
      CALL DEC (M, M, A, IP, IER)
      IF (IER .NE. 0) GO TO 300
      DO 70 J = 1,M
 70     CALL SOL (M, M, A, E(1,J,2), IP)
      CALL SOL (M, M, A, X(1,2), IP)
```

```
C MAIN LOOP.  READ AND PROCESS BLOCK-ROWS 3 TO N-1. -------------------
      DO 150 K = 3,NM1
         KM1 = K - 1
         CALL BLOX (M, K, A, E(1,1,K), C, X(1,K))
         DO 100 J = 1,M
            DO 90 I = 1,M
               DP = 0.
               DO 80 L = 1,M
 80               DP = DP + C(I,L)*E(L,J,KM1)
 90            A(I,J) = A(I,J) - DP
 100        CONTINUE
      DO 120 I = 1,M
         DP = 0.
         DO 110 L = 1,M
 110        DP = DP + C(I,L)*X(L,KM1)
 120     X(I,K) = X(I,K) - DP
         CALL DEC (M, M, A, IP, IER)
         IF (IER .NE. 0) GO TO 300
         DO 130 J = 1,M
 130        CALL SOL (M, M, A, E(1,J,K), IP)
         CALL SOL (M, M, A, X(1,K), IP)
 150     CONTINUE
C READ AND PROCESS THE LAST BLOCK-ROW. ---------------------------------
      K = N
      CALL BLOX (M, N, A, B, C, X(1,N))
      DO 180 J = 1,M
         DO 170 I = 1,M
            DP = 0.
            DO 160 L = 1,M
 160           DP = DP + B(I,L)*E(L,J,NM2)
 170        C(I,J) = C(I,J) - DP
 180     CONTINUE
      DO 210 J = 1,M
         DO 200 I = 1,M
            DP = 0.
            DO 190 L = 1,M
 190           DP = DP + C(I,L)*E(L,J,NM1)
 200        A(I,J) = A(I,J) - DP
 210     CONTINUE
      DO 230 I = 1,M
         DP = 0.
         DO 220 L = 1,M
 220        DP = DP + C(I,L)*X(L,NM1) + B(I,L)*X(L,NM2)
 230     X(I,N) = X(I,N) - DP
      CALL DEC (M, M, A, IP, IER)
      IF (IER .NE. 0) GO TO 300
      CALL SOL (M, M, A, X(1,N), IP)
C BACKWARD SOLUTION SWEEP. ---------------------------------------------
      DO 260 KB = 1,NM2
         K = N - KB
         KP1 = K + 1
         DO 250 I = 1,M
            DP = 0.
            DO 240 L = 1,M
 240           DP = DP + E(I,L,K)*X(L,KP1)
 250        X(I,K) = X(I,K) - DP
 260     CONTINUE
C INCLUDE EXTRA TERMS IN LAST BACKWARD STEP. ---------------------------
      DO 280 I = 1,M
         DP = 0.
         DO 270 L = 1,M
 270        DP = DP + E(I,L,1)*X(L,2) + E(I,L,N)*X(L,3)
 280     X(I,1) = X(I,1) - DP
      RETURN
C ERROR RETURNS. -------------------------------------------------------
 300  IER = K
      RETURN
 310  IER = -1
      RETURN
C----------------------- END OF SUBROUTINE BTMS ---------------------
      END
```

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $3.00

Page Range	Domestic Price	Page Range	Domestic Price
001–025	$ 3.50	326–350	10.00
026–050	4.00	351–375	10.50
051–075	4.50	376–400	10.75
076–100	5.00	401–425	11.00
101–125	5.50	426–450	11.75
126–150	6.00	451–475	12.00
151–175	6.75	476–500	12.50
176–200	7.50	501–525	12.75
201–225	7.75	526–550	13.00
226–250	8.00	551–575	13.50
251–275	9.00	576–600	13.75
276–300	9.25	601–up	*
301–325	9.75		

*Add $2.50 for each additional 100 page increment from 601 to 1,000 pages; add $4.50 for each additional 100 page increment over 1,000 pages.