ELSEVIER

Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT

Journal of Computational and Applied Mathematics 156 (2003) 179–200

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

Some algorithms for solving special tridiagonal block Toeplitz linear systems

Borislav V. Minchev

Department of Computer Science, University of Bergen, Bergen N-5020, Norway

Received 14 April 2002; received in revised form 25 October 2002

Abstract

This paper is focused on different methods and algorithms for solving tridiagonal block Toeplitz systems of linear equations. We consider the El-Sayed method (Ph.D. Thesis, 1996) for such systems and propose several modifications that lead to different algorithms, which we discuss in detail. Our algorithms are then compared with some classical techniques as far as implementation time is concerned, number of operations and storage. Comments and conclusions for computing efficiency of the proposed new algorithms are given. Numerical experiments corroborating the theoretical results are also presented.

MSC: 65F10

Keywords: Linear system; Block Toeplitz matrix; Matrix equation; Woodbury's formula

1. Introduction

Many problems arising in practice lead to the solution of linear systems of equations with special coefficient matrices. Tridiagonal block Toeplitz linear systems arise in numerical solution of ordinary and partial differential equations (ODE and PDE), interpolation problems, boundary value problems (BVP), etc. [2,3,7,13]. It is known that these systems have the form

$$Mx = f, \tag{1}$$

E-mail address: borko@ii.uib.no (B.V. Minchev).
URL: http://www.ii.uib.no/~borko/

0377-0427/03/$ - see front matter
doi:10.1016/S0377-0427(02)00911-1

where

$$M = \begin{pmatrix} A & B & & & & & \\ B^* & A & B & & & 0 & \\ & B^* & A & . & & & \\ & & . & . & . & & \\ & & & . & . & . & \\ & & & & . & . & \\ & 0 & & & . & . & B \\ & & & & & B^* & A \end{pmatrix}, \tag{2}$$

is an Hermitian tridiagonal block Toeplitz matrix with block size n. A and B are $m \times m$ matrices, x and f are column vectors with size nm.

The aim of this paper is to discuss different algorithms for solving (1) and compare them as far as time for implementation, number of operations and storage are concerned.

We organize the present paper as follows:

1. In Section 2 we review LU factorization, Cholesky factorization [8] and adaptation the Cyclic Reduction method [1] corresponding to the special form of M. We modify the method described in [4].
2. In Section 3 we develop algorithms based on these modifications in order to optimize floating point operations, memory space and implementation.
3. Finally, we verify the results in a number of numerical experiments.

2. Methods for solving special block tridiagonal Toeplitz linear system

Let us recall some classical direct methods for solving the linear system (1): LU factorization, Cholesky factorization, Cyclic Reduction [8], as well as one modification of LU factorization described in [4], which is based on the solution of a nonlinear matrix equation. For simplicity's we introduce the following notation $x = \{x_i\}_{i=1,\dots,n}$, $f = \{f_i\}_{i=1,\dots,n}$, where x_i and f_i are blocks with size $m \times 1$.

2.1. Block LU factorization

Matrix (2) admits the following LU factorization

$$M = LU,$$

where

$$L=\begin{pmatrix} A_1 & & & & & \\ B_1 & A_2 & & & 0 & \\ & B_2 & . & & & \\ & & . & . & & \\ & & & . & . & \\ & 0 & & & . & . \\ & & & & B_{n-1} & A_n \end{pmatrix}, \quad U=\begin{pmatrix} F_1 & G_1 & & & & \\ & F_2 & G_2 & & 0 & \\ & & . & . & & \\ & & & . & . & \\ & & & & . & . \\ & 0 & & & . & G_{n-1} \\ & & & & & F_n \end{pmatrix},$$

where by the matrices A_i, B_i, F_i and G_i satisfy the relations

$$\begin{aligned} &A_1F_1 = A, \\ &B_1 = B^*F_1^{-1}, \\ &G_1 = A_1^{-1}B, \\ &\left.\begin{aligned} &A_iF_i = A - B_{i-1}G_{i-1} \\ &B_i = B^*F_i^{-1} \\ &G_i = A_i^{-1}B \end{aligned}\right\} \quad \text{for } i = 2,\dots,n-1, \\ &A_nF_n = A - B_{n-1}G_{n-1}. \end{aligned} \tag{3}$$

The matrices A_i and F_i are lower and upper triangular, respectively, and are obtained by LU factorization.

Thus, solving the linear system (1) is equivalent to solving two simpler systems

$$Ly = f, \quad y = \{y_i\}_{i=1,\dots,n}$$

and

$$Ux = y, \quad x = \{x_i\}_{i=1,\dots,n},$$

whose solutions are

$$\begin{aligned} &y_1 = A_1^{-1}f_1, \\ &y_i = A_i^{-1}(f_i - B_{i-1}y_{i-1}) \quad \text{for } i = 2,\dots,n \end{aligned} \tag{4}$$

and

$$\begin{aligned} &x_n = F_n^{-1}y_n, \\ &x_i = F_i^{-1}(y_i - G_ix_{i+1}) \quad \text{for } i = n-1,\dots,1, \end{aligned} \tag{5}$$

respectively.

2.2. Block Cholesky factorization

When the matrix M is positive definite the following factorization

$$M = LL^*,$$

exists, where

$$L = \begin{pmatrix} A_1 & & & & \\ B_1 & A_2 & & 0 & \\ & B_2 & \ddots & & \\ & & \ddots & \ddots & \\ & 0 & & B_{n-1} & A_n \end{pmatrix}.$$

The matrices A_i, B_i satisfy the relations

$$\begin{aligned} &A_1 A_1^* = A, \\ &B_1 = B^*(A_1^*)^{-1}, \\ &\left.\begin{aligned} &A_i A_i^* = A - B_{i-1}B_{i-1}^* \\ &B_i = B^*(A_i^*)^{-1} \end{aligned}\right\} \quad \text{for } i = 2,\dots,n-1, \\ &A_n A_n^* = A - B_{n-1}B_{n-1}^*. \end{aligned} \tag{6}$$

It is well known that A_i and A_i^* are lower and upper triangular, respectively, and are obtained by Cholesky factorization.

In this manner, solving the linear system (1) is again equivalent to solving two simpler systems

$$Ly = f, \quad y = \{y_i\}_{i=1,\dots,n}$$

and

$$L^*x = y, \quad x = \{x_i\}_{i=1,\dots,n}.$$

The solution of the first system can be found by (4). The solution of the second system satisfies

$$\begin{aligned} &x_n = (A_n^*)^{-1} y_n, \\ &x_i = (A_i^*)^{-1}(y_i - B_i^* x_{i+1}) \quad \text{for } i = n-1,\dots,1. \end{aligned} \tag{7}$$

2.3. Block cyclic reduction

In this section we adapt Bini's method [1] to the special case when the coefficient matrix is given as in (2). Let us derive explicitly the substitution formulas for computing the block coordinates of

the solution x. Recall that block cyclic reduction can be applied only if the block size of M is power of 2, in other words, let $n = 2^p$. By performing an even–odd permutation of the block-rows and block columns in (1) we obtain

$$\begin{pmatrix} D_1^{(0)} & L^{(0)^*} \\ L^{(0)} & D_2^{(0)} \end{pmatrix} \begin{pmatrix} x_+^{(0)} \\ x_-^{(0)} \end{pmatrix} \begin{pmatrix} f_+^{(0)} \\ f_-^{(0)} \end{pmatrix}, \tag{8}$$

where

$$x_+^{(0)} = \{x_{+_k}^{(0)}\}_{k=1,\ldots,2^{p-1}}, \quad x_-^{(0)} = \{x_{-_k}^{(0)}\}_{k=1,\ldots,2^{p-1}}, \quad f_+^{(0)} = \{f_{+_k}^{(0)}\}_{k=1,\ldots,2^{p-1}},$$
$$f_-^{(0)} = \{f_{-_k}^{(0)}\}_{k=1,\ldots,2^{p-1}}$$

are column vectors, whose elements are blocks of size $m \times 1$, satisfying the relations

$$x_{+_k}^{(0)} = x_{2k}, \quad x_{-_k}^{(0)} = x_{2k-1}, \quad f_{+_k}^{(0)} = f_{2k}, \quad f_{-_k}^{(0)} = f_{2k-1} \quad \text{for } k = 1,\ldots,2^{p-1}.$$

The cells

$$D_1^{(0)} = D_2^{(0)} = \begin{pmatrix} A & & & \\ & \ddots & & 0 \\ & & \ddots & \\ 0 & & & A \end{pmatrix}, \quad L^{(0)} = \begin{pmatrix} B & & & \\ B^* & \ddots & & 0 \\ & \ddots & \ddots & \\ 0 & & \ddots & B \end{pmatrix}$$

are matrices of block size $2^{p-1} \times 2^{p-1}$.

We apply one step of block-Gaussian elimination to (8) and obtain

$$\begin{cases} (D_2^{(0)} - L^{(0)} D_1^{(0)^{-1}} L^{(0)^*}) x_-^{(0)} = f_-^{(0)} - L^{(0)} D_1^{(0)^{-1}} f_+^{(0)}, \\ x_+^{(0)} = D_1^{(0)^{-1}} (f_+^{(0)} - L^{(0)^*} x_-^{(0)}). \end{cases} \tag{9}$$

Let

$$M^{(1)} = D_2^{(0)} - L^{(0)} D_1^{(0)^{-1}} L^{(0)^*},$$
$$x^{(1)} = x_-^{(0)},$$
$$f^{(1)} = f_-^{(0)} - L^{(0)} D_1^{(0)^{-1}} f_+^{(0)}.$$

Note that now the first equation of (9) has the form

$$M^{(1)} x^{(1)} = f^{(1)}. \tag{10}$$

Observe, that the matrix $M^{(1)}$ has the form

$$M^{(1)} = \begin{pmatrix} F^{(1)} & B^{(1)} & & & & \\ B^{(1)^*} & A^{(1)} & . & & & \\ & . & . & . & & \\ & & . & . & . & \\ & & & . & . & B^{(1)} \\ & & & & B^{(1)^*} & A^{(1)} \end{pmatrix},$$

where $F^{(1)} = A - BA^{-1}B^*$. Obviously, it is also a block tridiagonal matrix and, except for the north-western corner block $F^{(1)}$ it has a block Toeplitz structure, with block size $2^{p-1}\times 2^{p-1}$. Applying once again an even–odd permutation of the block rows and block columns to (10), we obtain

$$\begin{pmatrix} D_1^{(1)} & L^{(1)^*} \\ L^{(1)} & D_2^{(1)} \end{pmatrix} \begin{pmatrix} x_+^{(1)} \\ x_-^{(1)} \end{pmatrix} \begin{pmatrix} f_+^{(1)} \\ f_-^{(1)} \end{pmatrix}, \tag{11}$$

where

$$x_+^{(1)} = \{x_{+_k}^{(1)}\}_{k=1,\dots,2^{p-2}}, \quad x_-^{(1)} = \{x_{-_k}^{(1)}\}_{k=1,\dots,2^{p-2}}, \quad f_+^{(1)} = \{f_{+_k}^{(1)}\}_{k=1,\dots,2^{p-2}},$$
$$f_-^{(1)} = \{f_{-_k}^{(1)}\}_{k=1,\dots,2^{p-2}}$$

are column vectors, whose elements are blocks of size $m \times 1$ and satisfy the relations

$$x_{+_k}^{(1)} = x_{2k}^{(1)}, \quad x_{-_k}^{(1)} = x_{2k-1}^{(1)}, \quad f_{+_k}^{(1)} = f_{2k}^{(1)}, \quad f_{-_k}^{(1)} = f_{2k-1}^{(1)} \quad \text{for } k = 1,\dots,2^{p-2}.$$

Again the cells

$$D_1^{(1)} = \begin{pmatrix} A^{(1)} & & & \\ & . & & 0 \\ & & . & \\ 0 & & & . \\ & & & A^{(1)} \end{pmatrix}, \quad D_2^{(1)} = \begin{pmatrix} F^{(1)} & & & \\ & A^{(1)} & & 0 \\ & & . & \\ 0 & & & . \\ & & & A^{(1)} \end{pmatrix},$$

$$L^{(1)} = \begin{pmatrix} B^{(1)} & & & \\ B^{(1)^*} & . & & 0 \\ & . & . & \\ 0 & & . & . \\ & & B^{(1)^*} & B^{(1)} \end{pmatrix}$$

are matrices of block size $2^{p-2} \times 2^{p-2}$. We apply again one step of Gaussian elimination to (11) and obtain

$$M^{(2)}x^{(2)} = f^{(2)},$$

where $M^{(2)}$ is of the same type as $M^{(1)}$, but with block size $2^{p-2} \times 2^{p-2}$. Proceeding in a similar fashion, we obtain a sequence of linear systems of the form

$$M^{(j)}x^{(j)} = f^{(j)} \quad \text{for } j = 1, \ldots, p,$$

where

$$M^{(j)} = \begin{pmatrix} F^{(j)} & B^{(j)} & & & & \\ B^{(j)^*} & A^{(j)} & . & & & \\ & . & . & . & & \\ & & . & . & . & \\ & & & . & . & B^{(j)} \\ & & & & B^{(j)^*} & A^{(j)} \end{pmatrix}$$

is square matrix with block of size 2^{p-j} for $j = 1, \ldots, p$. When $j = p$ the cells $M^{(p)} = F^{(p)}$. The blocks of the matrix $M^{(j)}$ obey the following relations:

$$\left.\begin{aligned} B^{(j)} &= -B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)}, \\ A^{(j)} &= A^{(j-1)} - B^{(j-1)^*}A^{(j-1)^{-1}}B^{(j-1)} - B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)^*}, \\ F^{(j)} &= F^{(j-1)} - B^{(j-1)}A^{(j-1)^{-1}}B^{(j-1)^*}, \end{aligned}\right\} \quad \text{for } j = 1, \ldots, p, \tag{12}$$

where $A^{(0)} = A$, $B^{(0)} = B$, $F^{(0)} = A$.

For the block column vectors $f^{(j)}$ and $x^{(j)}$, we have

$$\begin{aligned} &f^{(j)} = f_-^{(j-1)} - L^{(j-1)}D_1^{(j-1)}f_+^{(j-1)} && \text{for } j = 1, \ldots, p \\ &f_{+_k}^{(j)} = f_{2k}^{(j)}, \ f_{-_k}^{(j)} = f_{2k-1}^{(j)} \quad \text{for } k = 1, \ldots, 2^{p-j-1} && \text{for } j = 0, \ldots, p-1 \end{aligned} \tag{13}$$

and

$$\left.\begin{aligned} x_-^{(j-1)} &= x^{(j)}, \\ x_+^{(j-1)} &= D_1^{(j-1)^{-1}}(f_+^{(j-1)} - L^{(j-1)^*}x_-^{(j-1)}), \\ x_{2k}^{(j-1)} &= x_{+_k}^{(j-1)}, \ x_{2k-1}^{(j-1)} = x_{-_k}^{(j-1)} \quad \text{for } k = 1, \ldots, 2^{p-j-2}, \end{aligned}\right\} \quad \text{for } j = p, \ldots, 1, \tag{14}$$

where $f^{(0)} = f$, $x^{(p)} = F^{(p)^{-1}}f^{(p)}$, $x^{(0)} = x$.

The cells

$$D_1^{(j)} = \begin{pmatrix} A^{(j)} & & & \\ & \ddots & & 0 \\ & & \ddots & \\ 0 & & & A^{(j)} \end{pmatrix}, \quad L^{(j)} = \begin{pmatrix} B^{(j)} & & & \\ B^{(j)*} & \ddots & & 0 \\ & \ddots & \ddots & \\ 0 & & B^{(j)*} & B^{(j)} \end{pmatrix} \quad \text{for } j = 0, \ldots, p-1$$

are square matrices of block size 2^{p-j-1}.

2.4. A modification of LU factorization

In 1990 Rojo [14] proposed a new method for solving symmetric circulant tridiagonal linear systems and in recent years it has been modified to dial with matrices M having a special structure. For instance, in [5] it is adapted to the case when the coefficient matrix M is pentadiagonal and strongly diagonally dominant. In [11] M is allowed to be not diagonally dominant. El-Sayed [4] extended Rojo's method to tridiagonal block matrices. His approach consists in introducing a nonlinear matrix equation to solving problem (1). The algorithms we propose in this paper are based on [4] and investigate different approaches for solving the nonlinear matrix equation of El-Sayed. We discuss Woodbury's formula and its numerical implementation.

Firstly, let us describe an algorithm for solving parametric linear systems of the form.

$$Ny = f, \tag{15}$$

where

$$N = \begin{pmatrix} X & B & & & & \\ B^* & A & \ddots & & 0 & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & B \\ & 0 & & & B^* & A \end{pmatrix}$$

is a block tridiagonal matrix with block size n, X is a parameter block of size $m \times m$, and the vectors $y = \{y_i\}_{i=1,\ldots,n}$, and $f = \{f_i\}_{i=1,\ldots,n}$ are column vectors consisting of n blocks of size $m \times 1$.

The matrix N admits the following LU factorization

$$N = LU = \begin{pmatrix} I & & & \\ B^*X^{-1} & \ddots & & 0 \\ & \ddots & \ddots & \\ 0 & & B^*X^{-1} & I \end{pmatrix} \begin{pmatrix} X & B & & \\ & \ddots & \ddots & 0 \\ & & \ddots & \ddots & \\ 0 & & & B \\ & & & X \end{pmatrix},$$

where I is the $m \times m$ identity matrix. The above factorization exists when the parameter X satisfies the nonlinear matrix equation

$$X + B^*X^{-1}B = A. \tag{16}$$

Thus, solving the linear system (15) is equivalent to solving two simpler systems

$$\begin{aligned} &Lz = f, \quad z = \{z_i\}_{i=1,\ldots,n}, \\ &Uy = z, \quad y = \{y_i\}_{i=1,\ldots,n}, \end{aligned} \tag{17}$$

whose solutions are

$$\begin{aligned} &z_1 = f_1, \\ &z_i = f_i - B^*X^{-1}z_{i-1}, \quad i = 2,3,\ldots,n, \\ &y_n = X^{-1}z_n, \\ &y_i = X^{-1}(z_i - By_{i+1}), \quad i = n-1, n-2, \ldots, 1, \end{aligned} \tag{18}$$

respectively.

Now, we can find the solution of (1). The matrices M and N are related by relation

$$M = N + E_1V_1^{\mathrm{T}},$$

where

$$E_1 = \begin{pmatrix} I \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad V_1^{\mathrm{T}} = (A - X \quad 0 \quad \ldots \quad 0).$$

Using Woodbury's formula we have

$$M^{-1} = N^{-1} - N^{-1}E_1(I + V_1^{\mathrm{T}}N^{-1}E_1)^{-1}V_1^{\mathrm{T}}N^{-1}.$$

Therefore, the solution x of (1) is obtained from the vector y as follows:

$$\begin{aligned} x &= M^{-1}f \\ &= N^{-1}f - N^{-1}E_1(I + V_1^{\mathrm{T}}N^{-1}E_1)^{-1}V_1^{\mathrm{T}}N^{-1}f \\ &= y - N^{-1}E_1(I + (A-X)E_1^{\mathrm{T}}N^{-1}E_1)^{-1}V_1^{\mathrm{T}}y \\ &= y - N^{-1}E_1(I + (A-X)E_1^{\mathrm{T}}N^{-1}E_1)^{-1}(A-X)y_1. \end{aligned} \tag{19}$$

El-Sayed proposes following decomposition

$$N = LDV,$$

for computing $N^{-1}E_1$ and $E_1^{\mathrm{T}}N^{-1}E_1$, where

$$L = \begin{pmatrix} I & & & & \\ P & . & & & 0 \\ & . & . & & \\ & & & & \\ 0 & & . & . & \\ & & & P & I \end{pmatrix}, \quad V = \begin{pmatrix} I & Q & & & \\ & . & . & & 0 \\ & & & & \\ & & . & . & \\ 0 & & & . & Q \\ & & & & I \end{pmatrix},$$

$$D = \mathrm{diag}(X, X, \ldots, X)$$

are square matrices of block size n, $P = B^*X^{-1}$ and $Q = X^{-1}B$.

The matrix N^{-1} becomes

$$N^{-1} = V^{-1}D^{-1}L^{-1}.$$

If we denote $L^{-1} = (L_{ij})$ and $V^{-1} = (V_{ij})$, then

$$L_{ij} = \begin{cases} 0, & i < j \\ (-1)^{i-j}P^{i-j}, & i \geqslant j, \end{cases}$$

$$V_{ij} = \begin{cases} 0, & i > j \\ (-1)^{j-i}Q^{j-i}, & i \leqslant j, \end{cases}$$

$$D^{-1} = \mathrm{diag}(X^{-1}, X^{-1}, \ldots, X^{-1}).$$

Therefore, the blocks $(N^{-1}E_1)_i$ of the vector $N^{-1}E_1$ satisfy the formulas

$$\begin{aligned} (N^{-1}E_1)_i &= \left(V^{-1}D^{-1}L^{-1}E_1\right)_i \\ &= \sum_{s=i}^{n} (-1)^{i+1}Q^{s-i}X^{-1}P^{s-1} \quad \text{for } i = 1, \ldots, n. \end{aligned} \tag{20}$$

Hence

$$E_1^{\mathrm{T}} N^{-1} E_1 = (N^{-1}E_1)_1 = \sum_{s=1}^{n} Q^{s-1} X^{-1} P^{s-1}. \tag{21}$$

The coordinates x_i of the vector x in (19) are given by

$$x_i = y_i - \left[\sum_{s=i}^{n} (-1)^{i+1} Q^{s-i} X^{-1} P^{s-1}\right] \times \left[I + (A-X)\sum_{s=1}^{n} (-1)^{i+1} Q^{s-1} X^{-1} P^{s-1}\right]^{-1} (A-X) y_1 \quad \text{for } i = 1, \ldots, n.$$

Obviously, formulas (20) and (21) are not convenient to implement directly, because they require a great number of redundant multiplications. For this reason we propose two algorithms for computing the vector $N^{-1}E_1$.

Algorithm F. Solve m linear systems of type (17) with right-hand sides the corresponding to different columns of E_1. Note that this approach does not take into consideration the special structure of the right-hand sides vectors (having only a very sparse nonzero block). If the matrices A and B are real, the algorithm costs $O(8nm^3)$ flops and requires the storage of nm^2 real numbers.

Algorithm R. The blocks $(N^{-1}E_1)_i$ are recursively computed by formula (20) using the following algorithm:

- Find the cells $Y_i = (-1)^{n-i} X^{-1} P^{n-i}$ for $i = 1, \ldots, n$ by
  $$Y_1 = X^{-1},$$
  $$Y_i = Y_{i-1}(-P) \quad \text{for } i = 2, \ldots, n.$$
- Compute the blocks $(N^{-1}E_1)_i$ by
  $$(N^{-1}E_1)_n = Y_n,$$
  $$(N^{-1}E_1)_i = Y_i - Q(N^{-1}E_1)_{i+1} \quad \text{for } i = n-1 \div 1.$$

Our theoretical investigation shows that the algorithm R:

1. Requires half as many flops as the algorithm F, at the expense of minimal increase of storage memory. If the matrices A and B are real the algorithm costs $O(4nm^3)$ flops and needs to store $(n+2)m^2$ real numbers,
2. Takes advantage of the special form of the matrix E_1.

3. Algorithms for solving special block tridiagonal linear systems

In this section we compare the algorithms for solving special block tridiagonal linear systems described in Section 2 and there modifications.

3.1. Algorithm LU

1. *Find the matrices* A_i, B_i, F_i *and* G_i *according to* (3).
2. *Solve the system* $Ly = f$ *by* (4).
3. *Solve the system* $Ux = y$ *by* (5).
   end.

If the matrices A and B are real, this algorithm requires $O((15n/3)m^3 + (n/3)m^3 + (n/3)m^3) = O((17n/3)m^3)$ flops and the storage of $(3n+1)m^2 + 2nm$ real numbers.

3.2. Algorithm CHOL

1. *Find the matrices* A_i *and* B_i *according to* (6).
2. *Solve the system* $Ly = f$ *by* (4).
3. *Solve the system* $L^*x = y$ *by* (7).
   end.

If the matrices A and B are real, this algorithm requires $O((11n/3)m^3 + (n/3)m^3 + (n/3)m^3) = O((13n/3)m^3)$ flops and the storage of $[(3n+4)/2]m^2 + 3nm/2$ real numbers.

3.3. Algorithm CR (cyclic reduction)

1. *Find the matrices* $A^{(j)}, B^{(j)}$ *and* $F^{(j)}$ *for* $j = 1, \ldots, p$ *by* (12).
2. *Compute the vectors* $f^{(j)}$ *for* $j = 1, \ldots, p$ *by* (13).
3. *Solve the linear system* $F^{(p)}x^{(p)} = f^{(p)}$.
4. *Restore the coordinates of the vector* x *according to* (14).
   end.

If the matrices A and B are real, this algorithm requires $O([18p+2]m^3 + [4pm^3 + 4 * 2^p m^2] + 2m^3 + 6 * 2^p m^2) = O([22p+4]m^3 + 10 * 2^p m^2)$ flops and the storage of $(2p+9)m^2 + (8 * 2^p - 6)m$ real numbers, where $p = \log_2 n$.

Some identical computations are imposed by program realization on formulas (12) and (13). It is clear, that the vectors $f_-^{(j)}$ are not used in the next calculations. Based on this consideration and according to the special structure of the cells $D_1^{(j)}, D_2^{(j)}$ and $L^{(j)}$ the next new algorithm—modification of Cyclic Reduction is developed. It needs less flops and less storage.

3.4. Algorithm CRM (cyclic reduction-modification)

1. *Find the matrices* $A^{(j)}, B^{(j)}$, $F^{(j)}$ *and the vectors* $f_+^{(j)}$ *by the scheme*
   1.1. *Put*
   $A^{(0)} = A, B^{(0)} = B, F^{(0)} = F,$
   $f_{-_k} = f_{2k-1}$ for $k = 1, \ldots, 2^{p-1}$.

1.2. *For* $j = 1, 2, \ldots, p-1$ *compute*

$$
\begin{aligned}
&W_1 = B^{(j-1)}A^{(j-1)^{-1}},\\
&W_2 = B^{(j-1)^*}A^{(j-1)^{-1}},\\
&W_3 = W_1 B^{(j-1)^*},\\
&A^{(j)} = A^{(j-1)} - W_2 B^{(j-1)} - W_3,\\
&B^{(j)} = -W_1 B^{(j-1)},\\
&F^{(j)} = F^{(j-1)} - W_3,\\
&f_1 = f_{-_1} - W_1 f_{+_1}^{(j-1)},\\
&f_k = f_{-_k} - W_2 f_{+_{k-1}}^{(j-1)} - W_1 f_{+_k}^{(j-1)} \qquad \text{for } k = 2, \ldots, 2^{p-j},\\
&\left.\begin{aligned} f_{-_k} &= f_{2k-1}\\ f_{+_k}^{(j)} &= f_{2k}\end{aligned}\right\} \qquad \text{for} \quad k = 1, \ldots, 2^{p-j-1}.
\end{aligned}
$$

1.3. *Find*

$$
\begin{aligned}
&W_1 = B^{(p-1)}A^{(p-1)^{-1}},\\
&F^{(p)} = F^{(p-1)} - W_1 B^{(p-1)^*},\\
&f_1 = f_{-_1} - W_1 f_{+_1}^{(p-1)}.
\end{aligned}
$$

2. *Solve the linear system* $F^{(p)}x_1 = f_1$.

3. *Retrieve the coordinates of the vector* x *by the scheme*

$$
\left.\begin{aligned}
&x_2 = A^{(p-1)^{-1}}[f_{+_1}^{(p-1)} - B^{(p-1)^*}x_1],\\
&x_{+_k} = A^{(j-1)^{-1}}[f_{+_k}^{(j-1)} - B^{(j-1)^*}x_k - B^{(j-1)}x_{k+1}]\\
&\qquad \text{for } k = 1, \ldots, 2^{p-j} - 1\\
&x_{+_s} = A^{(j-1)^{-1}}[f_{+_s}^{(j-1)} - B^{(j-1)^*}x_s], \quad s = 2^{p-j}\\
&x_{2k-1} = x_k, \quad x_{2k} = x_{+_k} \quad \text{for } k = 2^{p-j}, \ldots, 1
\end{aligned}\right\} \quad j = p-1, \ldots, 1
$$

end.

If the matrices A and B are real, this algorithm requires $\mathrm{O}([12pm^3 + 4 * 2^p m^2] + 2m^3 + 6 * 2^p m^2) = \mathrm{O}([12p + 2]m^3 + 10 * 2^p m^2)$ flops and the storage of $(2p + 9)m^2 + (4 * 2^p - 1)m$ real numbers, where $p = \log_2 n$.

For the new algorithms based on the discussion in Section 2.4, we must address the problem of solving the nonlinear matrix equation (16). In [6], Engwerda proves that, if A is a positive-definite matrix, then solution of (16) is equivalent to the solution of following matrix equation

$$Z + \tilde{B}^* Z^{-1} \tilde{B} = I, \tag{22}$$

where $\tilde{B} = A^{-1/2}BA^{-1/2}$, $Z = A^{-1/2}XA^{-1/2}$. Thus, the results proposed in [6,10] can be readily adapted to produce following algorithms for (16).

Let $\varepsilon > 0$ be a fixed tolerance.

Algorithm EI_γ (Engwerda; Ivanov)

1. *Find the matrix* $\tilde{B} = A^{-1/2}BA^{-1/2}$.
2. *Solve Eq.* (22) *by the following algorithm*:

2.1. $Z_0 = \gamma I$, $(1/2 \leqslant \gamma \leqslant 1)$.
2.2. *For* $k = 1, 2, \ldots$ *compute*
$Z_{k+1} = I - \tilde{B}^* Z_k^{-1} \tilde{B}$,
if $\|Z_k - Z_{k+1}\|_\infty = \|Z_k + \tilde{B}^* Z_k^{-1} \tilde{B} - I\|_\infty \leqslant \varepsilon$, *then stop*
2.3. $Z_k \to Z_+$, *where* Z_+ *is maximal solution of* (22).
3. *Compute the solution* $X = A^{1/2} Z_{k+1} A^{1/2}$.
end.

If the matrices A and B are real, this algorithm requires $O(8m^3 + [2m^3 + 2m^3 + 2m^3]k + 4m^3) = O([6k + 12]m^3)$ flops, where k is number of iterations for solving (22). The algorithm needs to store $7m^2$ real numbers.

Since for each k, Z_k are positive definite matrices, can modify the above algorithms using the idea proposed by Zhan [15].

Algorithm EI_γM (Engwerda; Ivanov—Modification)

1. *Find the matrix* $\tilde{B} = A^{-1/2} B A^{-1/2}$.
2. *Solve Eq.* (22) *as follows*:
2.1. $Z_0 = \gamma I$, $\quad (\frac{1}{2} \leqslant \gamma \leqslant 1)$.
2.2. *For* $k = 1, 2, \ldots$
- *Compute the Cholesky factorization of* Z_k, $Z_k = \tilde{L}\tilde{L}^*$,
- *Solve the triangular matrix equation* $\tilde{L}\tilde{Z} = \tilde{B}$,
- *Compute* $Z_{k+1} = I - \tilde{Z}^* \tilde{Z}$,
if $\|Z_k - Z_{k+1}\|_\infty = \|Z_k + \tilde{B}^* Z_k^{-1} \tilde{B} - I\|_\infty \leqslant \varepsilon$, *then stop*
2.3. $Z_k \to Z_+$.
3. *Compute* $X = A^{1/2} Z_{k+1} A^{1/2}$.
end.

If the matrices A and B are real, this algorithm requires $O(8m^3 + [(m^3/3) + (4m^3/3) + 2m^3]k + 4m^3) = O([(11/3)k + 12]m^3)$ flops, where k is number of iterations. The algorithm stores $(17m^2 + m)/2$ real numbers.

In case we wish to solve (16) by a direct solver, we can use the following adaptation of the algorithm presented in [12].

Algorithm M (Meini)

1. *Set* $X_0 = A, A_0 = A, B_0 = B$.
2. *For* $k = 1, 2, \ldots$ *compute*
$W = A_k^{-1} B_k$,
$B_{k+1} = B_k W$,
$W = B_k^* W$,
$A_{k+1} = A_k - B_k A_k^{-1} B_k^* - W$,
$X_{k+1} = X_k - W$,
if $\|X_{k+1} - X_k\|_\infty \leqslant \varepsilon$ *for* $\varepsilon > 0$, *then stop.*
3. $X_k \to X_+$, *where* X_+ *is maximal solution of* (16).
end.

If the matrices A and B are real, this algorithm requires $O(2m^3 + 5 * 2m^3) = O(12m^3)$ flops per iteration and the storage of $7m^2$ real numbers.

In analogy with the algorithm EI_γM we can consider the following modification of Meini's algorithm.

Algorithm MM (Meini—Modification)

1. *Set* $X_0 = A, A_0 = A, B_0 = B$.
2. *For* $k = 1, 2, \ldots$
   2.1. *Compute the Cholesky factorization of* A_k, $A_k = \tilde{L}\tilde{L}^*$,
   2.2. *Solve the triangular matrix equations*
   $$\tilde{L}\tilde{Y} = B_k,$$
   $$\tilde{L}\tilde{Z} = B_k^*,$$
   2.3. *Compute*
   $$W = \tilde{Y}^*\tilde{Y},$$
   $$B_{k+1} = \tilde{Z}^*\tilde{Y},$$
   $$A_{k+1} = A_k - \tilde{Z}^*\tilde{Z} - W,$$
   $$X_{k+1} = X_k - W,$$
   if $\|X_k - X_{k+1}\|_\infty \leqslant \varepsilon$ *for* $\varepsilon > 0$, *then stop.*
3. $X_k \to X_+$.
   end.

If the matrices A and B are real, this algorithm requires $O((m^3/3)+(8m^3/3)+3*2m^3)=O(27m^3/3)$ flops, per iteration and the storage of $(19m^2 + m)/2$ real numbers.

The algorithms EI_γM and MM require less operations per iteration at the expense of a minimal increase of the memory space. However, their MATLAB implementation shows that they are slower than EI_γ and M, respectively. This is due because they call fewer built in MATLAB function.

The advantage of the algorithms M and MM is their quadratic convergence, which guarantees fewer iterations to reach the required accuracy.

Based on the discussion presented in this section, we propose the following new algorithm for (1).

3.5. Algorithm $\alpha(\beta)$

1. *Solve the matrix equation* (16) *by algorithm* α, *where* $\alpha \in \{EI\_\gamma, EI\_\gamma M, M, MM\}$.
2. *Find the vector* $y = N^{-1}f$ *by formulas* (18).
3. *Compute the matrix* $N^{-1}E_1$ *by algorithm* β, *where* $\beta \in \{F, R\}$.
4. *Compute the solution* x *of* (1) *by formula* (19) *with successive calculation of the expressions*:
   $C = (A - X)(N^{-1}E_1)_1$, $(I + C)^{-1}$, $(A - X)y_1$,
   $z = (I + C)^{-1}(A - X)y_1$, $x = y - N^{-1}E_1 z$.
   end.

For real matrices A and B, this algorithm requires $O(k * o_\alpha + 8nm^2 + o_\beta + 6m^3)$ flops and the storage of $m_\alpha + nm + m_\beta + (2m^2 + m)$ real numbers, where k is number of iterations for solving the matrix equation (16), o_α and m_α are, respectively, the number of operations and memory space required for implementation of the algorithm α. The numbers o_β and m_β are similarly defined.

4. Numerical experiments

In this section we wish to corroborate the discussion of Section 3 by solving (1), with M given as in (2), and exact solution $x = (1, 1, \ldots, 1)^T$.

In our numerical experiments, M is real, symmetric, with several block size n and several size and structure of the cells A and B. The above algorithms are compared by means of execution times and accuracy of the solution.

The codes are written in MATLAB language and computations are done on a PENTIUM computer. The results of the experiments are given in separate tables for each example. The following notation is used:

- LU stands for the LU algorithm.
- CHOL stands for the CHOL algorithm.
- CR stands for the CR algorithm.
- CRM stands for the CRM algorithm.
- EI_1(F) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI_γ with initial guess $Z_0 = I$, then $N^{-1}E_1$ is computed by algorithm F.
- EI_1(R) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI_γ with initial guess $Z_0 = I$, then $N^{-1}E_1$ is computed by algorithm R.
- EI_$\frac{1}{2}(R)$ stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm EI_γ with initial guess $Z_0 = \frac{1}{2}I$, then $N^{-1}E_1$ is computed by algorithm R.
- M(R) stands for the $\alpha(\beta)$ algorithm. The matrix equation (16) is solved by algorithm M, then $N^{-1}E_1$ is computed by algorithm R.
- $n = 2^p$ is the block size of matrix M and m is the size of each block.

For all programs the value of ε is set to $\varepsilon = 10^{-14}$.

- Iter is the smallest number k, for which

$$\|Z_k - Z_{k+1}\|_\infty \leqslant \varepsilon \quad \text{for algorithm EI\_}\gamma,$$

$$\|X_k - X_{k+1}\|_\infty \leqslant \varepsilon \quad \text{for algorithm M.}$$

- Err. $= \|x - \tilde{x}\|_\infty$, where $\tilde{x}$ is the computed solution.

Table 1 reports the flops and memory space required for each program.

From Table 1 we see that algorithm CRM requires less memory space than the others. Its number of operation depends of the relationship between the sizes m and n. If $m < 5n/6p$ then algorithm CRM requires $O(10nm^2)$ flops and if $m > 5n/6p$ then—$O(12m^3 \log_2 n)$. The algorithms EI_γ(R) and M(R) are more effective than the classical LU and CHOL. Even under the assumption that the

Table 1
Flops and memory space

Algorithm	Flops	Memory space
LU	$(17n/3)m^3 + O(nm^2)$	$3nm^2 + 2nm$
CHOL	$(13n/3)m^3 + O(nm^2)$	$(3n/2)m^2 + (3n/2)m$
CR	$22m^3 \log_2 n + 10nm^2 + O(nm)$	$2m^2 \log_2 n + 8nm$
CRM	$12m^3 \log_2 n + 10nm^2 + O(nm)$	$2m^2 \log_2 n + 4nm$
EI_γ(F)	$(8n + 6\text{Iter})m^3 + O(nm^2)$	$nm^2 + (n+1)m$
EI_γ(R)	$(4n + 6\text{Iter})m^3 + O(nm^2)$	$nm^2 + (n+1)m$
M(R)	$(4n + 12\text{Iter})m^3 + O(nm^2)$	$nm^2 + (n+1)m$

number of iterations Iter is considerably less than n, for $m \leqslant 3$ and $p \geqslant 3$ they require less flops than algorithm CRM.

In the next examples the cells A and B of the matrix M are chosen in such way that they guarantee the existence of a positive definite solution of Eq. (16) [9,12].

Example 1. The cells A and B are chosen like in Example 7.3 from [9], i.e.

$$A = \begin{pmatrix} 1.20 & -0.30 & 0.10 \\ -0.30 & 2.10 & 0.20 \\ 0.10 & 0.20 & 0.65 \end{pmatrix}, \quad B = \begin{pmatrix} 0.37 & 0.13 & 0.12 \\ -0.30 & 0.34 & 0.12 \\ 0.11 & -0.17 & 0.29 \end{pmatrix}.$$

Here $\|\tilde{B}\|_2 = \|A^{-1/2}BA^{-1/2}\|_2 = 0.511$. To reach the required accuracy in solving Eq. (16) Algorithm EI_1 needs 404 iterations while Algorithm M only 10 iterations. In Table 2 we present the execution time (in seconds) and the error, of each algorithm for different values of m and n.

Example 2. We let the cells of the matrix M to be the matrices of example 5.2 from [12], i.e. $A = I$, and $B = (b_{i,j})$ a symmetric matrix, whose entries are determined as below: by the scheme:

1. Fix a real $0 \leqslant \alpha \leqslant 1/2$.
2. For $i = 1, \ldots, m$
   For $j = i, \ldots, m$
   $b_{i,j} = 2 * i + j$
   end j.
   Compute $s_1 = \sum_{j=1}^{i-1} b_{i,j}, \quad s_2 = \sum_{j=i}^{m} b_{i,j}$.
   For $j = i, \ldots, m$
   $b_{i,j} = \frac{b_{i,j}(1/2-\alpha-s_1)}{s_2}, \quad b_{j,i} = b_{i,j}$
   end j.

end i.

Table 2
Execution time (in seconds) and errors for Example 1

Algorithm	$m = 3$			
	$n = 2^6 = 64$		$n = 2^{10} = 1024$	
	Err.	Time	Err.	Time
LU	2.0650e − 014	0.06	3.3640e − 014	1.04
CHOL	1.3545e − 014	0.05	5.1292e − 014	0.88
CR	1.2079e − 013	0.05	3.2019e − 013	1.42
CRM	6.1284e − 014	0.06	1.6398e − 013	0.77
EI_1(F)	1.4211e − 014	0.16	8.4155e − 014	1.81
EI_1(R)	1.9540e − 014	0.11	8.4155e − 014	0.83
M(R)	1.2879e − 014	0.06	5.8620e − 014	0.77
	$n = 2^8 = 256$		$n = 2^{12} = 4096$	
LU	2.8089e − 014	0.27	3.3640e − 014	5.99
CHOL	4.6407e − 014	0.27	5.1292e − 014	4.45
CR	3.0931e − 013	0.33	3.2019e − 013	11.32
CRM	1.5998e − 013	0.22	1.6398e − 013	3.40
EI_1(F)	2.9532e − 014	0.55	3.0065e − 013	9.72
EI_1(R)	2.9976e − 014	0.28	3.0065e − 013	4.34
M(R)	3.2863e − 014	0.22	2.5757e − 013	4.23

The matrix B is symmetric, nonnegative and such that $Be = (1/2 - \alpha)e$, where e is the vector having all its entries equal to 1. Thus $\|\tilde{B}\|_2 = \|A^{-1/2}BA^{-1/2}\|_2 = 1/2 - \alpha$.

We consider the following two cases:

1. $\alpha = 0.4$; then $\|\tilde{B}\|_2 = 0.1$. To obtain the required accuracy for (16), Algorithm EI_1 needs 8 iterations, Algorithm M 4 iterations.
2. $\alpha = 0$; then $\|\tilde{B}\|_2 = \frac{1}{2}$. In this case, we solve Eq. (22) by algorithm EI_γ, with initial guess $Z_0 = \frac{1}{2}I$ ($\gamma = \frac{1}{2}$). To reach the required accuracy for (16), Algorithm EI_$\frac{1}{2}$ needs 9 iterations. The use of Algorithm EI_1 is not recommended, because it needs more than $3 \cdot 10^6$ iterations. Algorithm M needs of 32 iterations.

In Tables 3 and 4 we give execution time (in seconds) and errors, for $\alpha = 0$ and $\alpha = 0.4$, respectively.

Example 3. Let

$$A = \mathrm{circ}(20, -8, 1, \ldots, 1, -8),$$

be a circulant matrix and $B = I$. In this case $\|\tilde{B}\|_2 = \|A^{-1/2}BA^{-1/2}\|_2 = 0.1667$. To reach the required accuracy for (16) Algorithm EI_1 needs 10 iterations, Algorithm M—5 iterations.

Table 3
Execution time (in seconds) and errors for Example 2: ($\alpha = 0$)

Algorithm	$m = 3$		$m = 5$		$m = 10$	
	Err.	Time	Err.	Time	Err.	Time
$n = 2^6 = 64$						
LU	8.8818e − 015	0.11	4.3299e − 015	0.11	1.3989e − 014	0.11
CHOL	1.1990e − 014	0.06	7.7716e − 015	0.06	3.9968e − 015	0.11
CR	6.3505e − 014	0.05	4.9960e − 015	0.05	7.6605e − 015	0.11
CRM	4.3521e − 014	0.06	1.6431e − 014	0.06	2.2204e − 015	0.06
EI$_{-\frac{1}{2}}$(R)	6.3949e − 014	0.05	7.1054e − 014	0.06	8.5265e − 014	0.06
M(R)	1.7764e − 013	0.05	1.5632e − 013	0.05	2.8422e − 014	0.06
$n = 2^8 = 256$						
LU	5.4845e − 014	0.22	2.7645e − 014	0.28	8.7153e − 014	0.55
CHOL	1.3767e − 013	0.22	5.1958e − 014	0.28	6.4615e − 014	0.44
CR	1.0281e − 012	0.27	4.8628e − 014	0.27	9.7033e − 014	0.33
CRM	6.8057e − 013	0.16	2.4425e − 013	0.22	4.0079e − 014	0.22
EI$_{-\frac{1}{2}}$(R)	1.0658e − 012	0.17	4.2633e − 012	0.22	2.5295e − 012	0.22
M(R)	5.6843e − 013	0.17	8.5265e − 013	0.22	7.1054e − 013	0.22
$n = 2^{10} = 1024$						
LU	3.9635e − 013	1.10	6.2017e − 013	1.26	2.7101e − 013	2.58
CHOL	3.6748e − 013	0.88	1.4135e − 012	1.04	3.4261e − 013	2.03
CR	1.6434e − 011	1.42	6.9700e − 013	1.44	1.3676e − 012	1.48
CRM	1.0729e − 011	0.82	3.9986e − 012	0.83	6.7812e − 013	0.88
EI$_{-\frac{1}{2}}$(R)	3.8426e − 011	0.71	6.7075e − 012	0.75	3.6380e − 011	1.04
M(R)	3.0809e − 011	0.73	2.0236e − 011	0.77	8.1968e − 011	1.05
$n = 2^{12} = 4096$						
LU	5.5140e − 012	5.99	4.9095e − 012	6.70	4.1102e − 012	19.17
CHOL	8.9115e − 012	4.45	4.0730e − 012	5.05	2.1547e − 012	11.32
CR	2.6242e − 010	11.73	1.1125e − 011	11.87	2.1823e − 011	11.98
CRM	1.7121e − 010	3.35	6.3752e − 011	3.40	1.0165e − 011	3.51
EI$_{-\frac{1}{2}}$(R)	1.0141e − 009	4.17	9.7543e − 010	4.34	5.6480e − 010	5.77
M(R)	6.2664e − 010	4.23	4.4201e − 010	4.34	7.0941e − 010	5.76

In Table 5 we give the execution time (in seconds) and the error of each algorithms for different values of m and n.

5. Conclusions

From the discussion and the results obtained by numerical experiments, we can conclude that:

1. The proposed modifications of formulas (20) and (21) (Algorithm R) lead to a considerable decrease in the number of operations, for computing the block vector $N^{-1}E_1$. That explains

Table 4
Execution time (in seconds) and errors for Example 2: ($\alpha = 0.4$)

Algorithm	$m = 3$		$m = 5$		$m = 10$	
	Err.	Time	Err.	Time	Err.	Time
$n = 2^6 = 64$						
LU	4.4409e − 016	0.05	4.4409e − 016	0.11	8.8818e − 016	0.11
CHOL	6.6613e − 016	0.05	5.5511e − 016	0.06	8.8818e − 016	0.11
CR	4.4409e − 016	0.05	5.5511e − 016	0.05	6.6613e − 016	0.06
CRM	4.4409e − 016	0.05	5.5511e − 016	0.05	6.6613e − 016	0.06
EI_1(F)	6.6613e − 016	0.11	3.3307e − 016	0.17	08.8818e − 016	0.33
EI_1(R)	6.6613e − 016	0.05	3.3307e − 016	0.06	8.8818e − 016	0.06
M(R)	4.4409e − 016	0.05	5.5511e − 016	0.05	7.7716e − 016	0.06
$n = 2^8 = 256$						
LU	4.4409e − 016	0.21	4.4409e − 016	0.27	8.8818e − 016	0.55
CHOL	6.6613e − 016	0.22	5.5511e − 016	0.27	8.8818e − 016	0.44
CR	4.4409e − 016	0.27	5.5511e − 016	0.28	6.6613e − 016	0.28
CRM	4.4409e − 016	0.18	5.5511e − 016	0.20	6.6613e − 016	0.22
EI_1(F)	6.6613e − 016	0.44	3.3307e − 016	0.60	8.8818e − 016	1.16
EI_1(R)	6.6613e − 016	0.17	3.3307e − 016	0.22	8.8818e − 016	0.22
M(R)	4.4409e − 016	0.16	5.5511e − 016	0.16	7.7716e − 016	0.22
$n = 2^{10} = 1024$						
LU	4.4409e − 016	1.04	4.4409e − 016	1.27	8.8818e − 016	2.59
CHOL	6.6613e − 016	0.94	5.5511e − 016	1.05	8.8818e − 016	1.97
CR	4.4409e − 016	1.43	5.5511e − 016	1.42	6.6613e − 016	1.48
CRM	4.4409e − 016	0.77	5.5511e − 016	0.88	6.6613e − 016	0.83
EI_1(F)	6.6613e − 016	1.76	3.3307e − 016	2.52	8.8818e − 016	4.78
EI_1(R)	6.6613e − 016	0.76	3.3307e − 016	0.83	8.8818e − 016	1.10
M(R)	4.4409e − 016	0.72	5.5511e − 016	0.77	7.7716e − 016	0.99
$n = 2^{12} = 4096$						
LU	4.4409e − 016	5.99	4.4409e − 016	6.70	8.8818e − 016	18.51
CHOL	6.6613e − 016	4.45	5.5511e − 016	5.11	8.8818e − 016	10.93
CR	4.4409e − 016	11.81	5.5511e − 016	12.14	6.6613e − 016	12.31
CRM	4.4409e − 016	3.35	5.5511e − 016	3.40	6.6613e − 016	3.57
EI_1(F)	6.6613e − 016	9.62	3.3307e − 016	13.74	8.8818e − 016	24.33
EI_1(R)	6.6613e − 016	4.22	3.3307e − 016	4.28	8.8818e − 016	5.99
M(R)	4.4409e − 016	4.23	5.5511e − 016	4.34	7.7716e − 016	5.76

why the execution time for Algorithms EI_γ(R) and M(R) is less than for Algorithm EI_γ(F).

2. The adapted algorithms CRM, EI_γ(R) and M(R) essentially take advantage of the special structure of the matrix M, and this makes them more effective than the classical algorithms LU, CHOL, CR as far as the number of operation, memory requirements and execution time are concerned.

Table 5
Execution time (in seconds) and errors for Example 3

Algorithm	$m=5$		$m=7$		$m=10$	
	Err.	Time	Err.	Time	Err.	Time
$n=2^6=64$						
LU	6.6613e − 016	0.05	6.6613e − 016	0.05	1.3323e − 015	0.11
CHOL	1.1102e − 015	0.05	6.6613e − 016	0.05	8.8818e − 016	0.11
CR	6.6613e − 016	0.06	1.1102e − 015	0.06	1.7764e − 015	0.06
CRM	7.7716e − 016	0.05	1.7764e − 015	0.06	1.5543e − 015	0.06
EI_1(F)	3.7748e − 015	0.11	2.4425e − 015	0.22	2.8866e − 015	0.33
EI_1(R)	3.7748e − 015	0.05	2.4425e − 015	0.06	2.8866e − 015	0.06
M(R)	5.5511e − 016	0.05	4.4409e − 016	0.05	6.6613e − 016	0.06
$n=2^8=256$						
LU	6.6613e − 016	0.33	6.6613e − 016	0.38	1.3323e − 015	0.50
CHOL	1.1102e − 015	0.22	6.6613e − 016	0.33	8.8818e − 016	0.44
CR	6.6613e − 016	0.27	1.1102e − 015	0.28	1.7764e − 015	0.28
CRM	7.7716e − 016	0.17	1.7764e − 015	0.22	1.5543e − 015	0.22
EI_1(F)	3.7748e − 015	0.60	2.4425e − 015	0.83	2.8866e − 015	1.15
EI_1(R)	3.7748e − 015	0.22	2.4425e − 015	0.22	2.8866e − 015	0.23
M(R)	5.5511e − 016	0.16	4.4409e − 016	0.17	6.6613e − 016	0.27
$n=2^{10}=1024$						
LU	6.6613e − 016	1.26	6.6613e − 016	1.82	1.3323e − 015	2.59
CHOL	1.1102e − 015	1.05	6.6613e − 016	1.21	8.8818e − 016	1.98
CR	6.6613e − 016	1.48	1.1102e − 015	1.42	1.7764e − 015	1.48
CRM	7.7716e − 016	0.82	1.7764e − 015	0.88	1.5543e − 015	0.88
EI_1(F)	3.7748e − 015	2.58	2.4425e − 015	3.46	2.8866e − 015	4.83
EI_1(R)	3.7748e − 015	0.77	2.4425e − 015	0.83	2.8866e − 015	1.10
M(R)	5.5511e − 016	0.77	4.4409e − 016	0.82	6.6613e − 016	0.99
$n=2^{12}=4096$						
LU	6.6613e − 016	6.70	6.6613e − 016	7.58	1.3323e − 015	18.290
CHOL	1.1102e − 015	5.05	6.6613e − 016	5.71	8.8818e − 016	11.15
CR	6.6613e − 016	11.84	1.1102e − 015	11.91	1.7764e − 015	12.58
CRM	7.7716e − 016	3.69	1.7764e − 015	3.74	1.5543e − 015	3.87
EI_1(F)	3.7748e − 015	13.89	2.4425e − 015	18.29	2.8866e − 015	24.71
EI_1(R)	3.7748e − 015	4.45	2.4425e − 015	4.67	2.8866e − 015	5.93
M(R)	5.5511e − 016	4.40	4.4409e − 016	4.62	6.6613e − 016	5.77

3. The Algorithms CRM, EI_γ(R) and M(R) are comparable for accuracy of the computed solution, execution time for required flops (for $m \leqslant 3$ and $p \geqslant 3$). For large values of the m, Algorithm CRM requires the least flops, which, together with the fact that it uses the least memory space, suggests that it is most suitable when the block size of M is a power of two.
4. If n is not a power of two and the cells of the matrix M satisfy the conditions for existence of the solution of Eq. (16), then the use of Algorithms M(R) is recommended instead.

Acknowledgements

This work was partially supported by Shoumen University under contract N 17/2001. I wish to thank Ivan Ivanov and Antonella Zanna for their helpful remarks and comments.

References

[1] D. Bini, B. Meini, Solving block banded block Toeplitz systems with structured blocks: new algorithms and open problems, in: Large-scale Scientific Computations of Engineering and Environmental Problems II, Notes on Numerical Fluid Mechanics, Vol. 73, Vieweg, Braunschweig, Wiesbaden, 2000, pp. 15–24.
[2] B.L. Buzbee, G.H. Golub, C.W. Nielson, On direct methods for solving Poisson's equations, SIAM J. Numer. Anal. 7 (1970) 627–656.
[3] F. Diele, L. Lopez, The use of the factorization of five-diagonal matrices by tridiagonal Toeplitz matrices, Appl. Math. Lett. 11 (1998) 61–69.
[4] S.M. El-Sayed, Investigation of special matrices and numerical methods for special matrix equations. Ph.D. Thesis, Sofia, 1996 (in Bulgarian).
[5] S.M. El-Sayed, I.G. Ivanov, M.G. Petkov, A new modification of the Rojo Method for solving symmetric circulant five-diagonal systems of linear equations, Comput. Math. Appl. 35 (1998) 35–44.
[6] J.C. Engwerda, C.M. Ran Andre, A.L. Rijkeboer, Necessary and sufficient conditions for the existence of a positive definite solution of the matrix equation $X + A^*X^{-1}A = Q$, Linear Algebra Appl. 186 (1993) 255–275.
[7] G. Fiorentino, S. Serra, Multigrid methods for symmetric positive definite block Toeplitz matrices with nonnegative generating functions, SIAM J. Sci. Comput. 17 (1996) 1068–1081.
[8] G. Golub, C. Van Loan, Matrix Computation, The Johns Hopkins University Press, Baltimore, MD, 1989.
[9] C.-H. Guo, P. Lancaster, Iterative solution of two matrix equations, Math. Comput. 68 (1999) 1589–1603.
[10] I. Ivanov, V. Hasanov, F. Uhlig, Iterative methods for computing a positive definite solution of matrix equations $X \pm A^*X^{-1}A = I$, Math. Comput., submitted.
[11] I. Ivanov, B. Mintchev, A method for solving special circulant pentadiagonal linear system, in: Large-scale Scientific Computations of Engineering and Environmental Problems II, Notes on Numerical Fluid Mechanics, Vol. 73, Vieweg, Braunschweig, Wiesbaden, 2000, pp. 144–151.
[12] B. Meini, Matrix Equations and Structures: Efficient Solution of Special Discrete Algebraic Riccati Equations, in: Second Conference on Numerical Analysis and Applications, Lecture Notes in Computer Science 1988, Springer, Berlin, 2000, pp. 578–585. http://link.springer.de/link/service/series/0558/tocs/t1988.htm
[13] J. Rissanen, Solution of linear equations with Hancel and Toeplitz matrices, Numer. Math. 22 (1974) 361–366.
[14] O. Rojo, A new method for solving symmetric circulant tridiagonal systems of linear equations, Comput. Math. Appl. 20 (1990) 61–67.
[15] X. Zhan, Computing the extremal positive definite solutions of a matrix equation, SIAM J. Sci. Comput. 17 (1996) 1167–1174.