DIRECT METHODS FOR MATRIX SYLVESTER AND LYAPUNOV EQUATIONS

DANNY C. SORENSEN AND YUNKAI ZHOU

Received 12 December 2002 and in revised form 31 January 2003

We revisit the two standard dense methods for matrix Sylvester and Lyapunov equations: the Bartels-Stewart method for $\mathbf{A}_1\mathbf{X}+\mathbf{X}\mathbf{A}_2+\mathbf{D}=\mathbf{0}$ and Hammarling's method for $\mathbf{AX}+\mathbf{XA}^T+\mathbf{BB}^T=\mathbf{0}$ with $\mathbf{A}$ stable. We construct three schemes for solving the unitarily reduced quasitriangular systems. We also construct a new rank-1 updating scheme in Hammarling's method. This new scheme is able to accommodate a $\mathbf{B}$ with more columns than rows as well as the usual case of a $\mathbf{B}$ with more rows than columns, while Hammarling's original scheme needs to separate these two cases. We compared all of our schemes with the Matlab Sylvester and Lyapunov solver *lyap.m*; the results show that our schemes are much more efficient. We also compare our schemes with the Lyapunov solver *sllyap* in the currently possibly the most efficient control library package SLICOT; numerical results show our scheme to be competitive.

1. Introduction

Matrix Sylvester equation and Lyapunov equation are very important in control theory and many other branches of engineering. We discuss dense methods for the Sylvester equation

$$\mathbf{A}_1\mathbf{X}+\mathbf{X}\mathbf{A}_2+\mathbf{D}=\mathbf{0}, \tag{1.1}$$

where $\mathbf{A}_1\in\mathbb{R}^{m\times m}$, $\mathbf{A}_2\in\mathbb{R}^{n\times n}$, and $\mathbf{D}\in\mathbb{R}^{m\times n}$; and the Lyapunov equation

Copyright © 2003 Hindawi Publishing Corporation
Journal of Applied Mathematics 2003:6 (2003) 277–303
2000 Mathematics Subject Classification: 15A24, 15A06, 65F30, 65F05
URL: http://dx.doi.org/10.1155/S1110757X03212055

$$\mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{A}^T + \mathbf{B}\mathbf{B}^T = \mathbf{0}, \tag{1.2}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{A}$ is stable, and $\mathbf{B} \in \mathbb{R}^{n\times p}$.

The Bartels-Stewart method [3] has been the method of choice for solving small-to-medium scale Sylvester and Lyapunov equations. And for about twenty years, Hammarling's paper [11] remains the one and only reference for directly computing the Cholesky factor of the solution $\mathbf{X}$ of (1.2) for small to medium n.

Many matrix equations that arise in practice are small to medium scale. Moreover, most projection schemes for large scale matrix equations require efficient and stable direct methods to solve the small-to-medium scale projected matrix equations. Therefore, it is worthwhile to revisit existing dense methods and make performance improvements.

In Section 1, we briefly review the Bartels-Stewart method for (1.1), this method is implemented as *lyap.m* in Matlab; we introduce three modified schemes for solving the reduced quasitriangular Sylvester equation in the real case. Numerical evidence indicates that our schemes are much more efficient than the Matlab function *lyap.m*. In Section 2, we revisit Hammarling's method for (1.2), again for the reduced triangular Lyapunov equation. We present our new updating formulations of the Cholesky factor of $\mathbf{X}$ for (1.2) both in complex arithmetic and in real arithmetic. We give computational evidence that our methods are much more efficient than the Matlab function *lyap.m*. We also compare our method with possibly the most efficient Lyapunov solver currently available: *sllyap* in the control library package SLICOT [4] (see http://www.win.tue.nl/niconet/NIC2/slicot.html); numerical results show our formulations to be competitive.

For other applications of triangular Sylvester and Lyapunov equations and recent recursive methods, see the eloquent and beautifully written paper [14]. Numerical methods for generalized or coupled matrix equations can be found in [5, 14, 16].

2. Bartels-Stewart algorithm and its improvements in the real case

The Bartels-Stewart method provided the first numerically stable way to systematically solve the Sylvester equation (1.1). For the backward stability analysis of Bartels-Stewart algorithm, see [12] and [13, page 313]. Another backward error analysis can be found in [7]. The Bartels-Stewart algorithm is now standard and is presented in textbooks (e.g., [10, 17]).

The main idea of the Bartels-Stewart algorithm is to apply the Schur decomposition [10] to transform (1.1) into a triangular system which can be solved efficiently by forward or backward substitutions. Our formulation for the real case uses the same idea.

Throughout, we assume that $\mathbf{A}_1$ and $-\mathbf{A}_2$ have no common eigenvalues. This is a necessary and sufficient condition for (1.1) to have a unique solution.

2.1. $\mathbf{A}_2$ *is transformed into upper quasitriangular matrix*

We first discuss the case when $\mathbf{A}_2$ is transformed into (quasi-) upper triangular matrix. The other cases will be discussed in the next section.

Let the Schur decomposition of $\mathbf{A}_1$, $\mathbf{A}_2$ be

$$\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1^H, \qquad \mathbf{A}_2 = \mathbf{Q}_2\mathbf{R}_2\mathbf{Q}_2^H, \tag{2.1}$$

where $\mathbf{Q}_1$, $\mathbf{Q}_2$ are unitary and $\mathbf{R}_1$, $\mathbf{R}_2$ are upper triangular. Then (1.1) is equivalent to

$$\mathbf{R}_1\tilde{\mathbf{X}} + \tilde{\mathbf{X}}\mathbf{R}_2 + \tilde{\mathbf{D}} = \mathbf{0}, \quad \text{where } \tilde{\mathbf{D}} = \mathbf{Q}_1^H\mathbf{D}\mathbf{Q}_2. \tag{2.2}$$

Once $\tilde{\mathbf{X}}$ is solved, the solution $\mathbf{X}$ of (1.1) can be obtained by $X = \mathbf{Q}_1\tilde{X}\mathbf{Q}_2^H$.

Denote

$$\begin{aligned}\tilde{\mathbf{X}} &= \left[\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_n\right], \quad \tilde{\mathbf{x}}_i \in \mathbb{R}^m, \\ \tilde{\mathbf{D}} &= \left[\tilde{\mathbf{d}}_1, \tilde{\mathbf{d}}_2, \ldots, \tilde{\mathbf{d}}_n\right], \quad \tilde{\mathbf{d}}_i \in \mathbb{R}^m.\end{aligned} \tag{2.3}$$

Denote $\mathbf{R}_2 = [r_{ij}^{(2)}]$, $i,j = 1,\ldots,n$. By comparing columns in (2.2), we obtain the formula for the columns of $\tilde{\mathbf{X}}$,

$$\left(\mathbf{R}_1 + r_{kk}^{(2)}\mathbf{I}\right)\tilde{\mathbf{x}}_k = -\tilde{\mathbf{d}}_k - \sum_{i=1}^{k-1}\tilde{\mathbf{x}}_i r_{ik}^{(2)}, \quad k = 1,2,\ldots,n. \tag{2.4}$$

Since $\Lambda(\mathbf{A}_1) \cap \Lambda(\mathbf{A}_2) = \emptyset$, for each k, (2.4) always has a unique solution. We can solve (2.2) by forward substitutions, that is, solving $\tilde{\mathbf{x}}_1$ first, putting it into the right-hand side of (2.4), then we can solve for $\tilde{\mathbf{x}}_2$. All $\tilde{\mathbf{x}}_3, \ldots, \tilde{\mathbf{x}}_n$ can be solved sequentially. The Matlab function *lyap.m* implements this complex version of the Bartels-Stewart algorithm.

When the coefficient matrices are real, it is possible to restrict computation to real arithmetic as shown in [3]. A discussion of this may also be found in Golub and Van Loan [10]. Both of these variants resort to the equivalence relation between (2×2)-order Sylvester equations and (4×1)-order linear systems via Kronecker product when there are complex conjugate eigenvalues.

We introduce three columnwise direct solve schemes which do not use the Kronecker product. Our column-wise block solve scheme is more suitable for a modern computer architecture where cache performance is important. The numerical comparison results done in the Matlab environment show that our schemes are much more efficient than the complex version *lyap.m*.

We first perform real Schur decomposition of $\mathbf{A}_1$, $\mathbf{A}_2$:

$$\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1^T, \qquad \mathbf{A}_2 = \mathbf{Q}_2\mathbf{R}_2\mathbf{Q}_2^T. \tag{2.5}$$

Note now that all matrices in (2.5) are real, $\mathbf{Q}_1$, $\mathbf{Q}_2$ are orthogonal matrices, and $\mathbf{R}_1$, $\mathbf{R}_2$ are now real upper quasitriangular, with diagonal block-size 1 or 2 where block of size 2 corresponds to a conjugate pair of eigenvalues.

Equation (2.4) implies that we only need to look at the diagonal blocks of $\mathbf{R}_2$. When the kth diagonal block of $\mathbf{R}_2$ is of size 1 (i.e., $r_{k+1,k}^{(2)} = 0$), we can apply the same formula as (2.4) to get x_k; when the size of the kth diagonal block is of size 2 (i.e., $r_{k+1,k}^{(2)} \neq 0$), we need to solve for both $\tilde{\mathbf{x}}_k$, $\tilde{\mathbf{x}}_{k+1}$ at the kth step and then move to the $(k+2)$th block.

Our *column-wise elimination scheme* for size-2 diagonal block proceeds as follows: denote

$$\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} := \begin{bmatrix} r_{k,k}^{(2)} & r_{k,k+1}^{(2)} \\ r_{k+1,k}^{(2)} & r_{k+1,k+1}^{(2)} \end{bmatrix}. \tag{2.6}$$

By comparing the $\{k, k+1\}$th columns in (2.2), we get

$$\mathbf{R}_1\left[\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}\right] + \left[\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}\right]\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = -\left[\tilde{\mathbf{d}}_k, \tilde{\mathbf{d}}_{k+1}\right] - \left[\sum_{i=1}^{k-1}\tilde{\mathbf{x}}_i r_{i,k}^{(2)}, \sum_{i=1}^{k-1}\tilde{\mathbf{x}}_i r_{i,k+1}^{(2)}\right]. \tag{2.7}$$

Since at the kth step $\{\mathbf{x}_i,\ i = 1,\ldots,k-1\}$ are already solved, the two columns in the right-hand side of (2.7) are known vectors, we denote them as $[\mathbf{b}_1, \mathbf{b}_2]$. Then by comparing columns in (2.7), we get

$$\mathbf{R}_1\tilde{\mathbf{x}}_k + r_{11}\tilde{\mathbf{x}}_k + r_{21}\tilde{\mathbf{x}}_{k+1} = \mathbf{b}_1, \tag{2.8}$$

$$\mathbf{R}_1\tilde{\mathbf{x}}_{k+1} + r_{12}\tilde{\mathbf{x}}_k + r_{22}\tilde{\mathbf{x}}_{k+1} = \mathbf{b}_2. \tag{2.9}$$

From (2.8), we get

$$\tilde{\mathbf{x}}_{k+1} = \frac{1}{r_{21}}\left\{\mathbf{b}_1 - \left(\mathbf{R}_1 + r_{11}\mathbf{I}\right)\tilde{\mathbf{x}}_k\right\}. \tag{2.10}$$

From (2.9),

$$\left(\mathbf{R}_1 + r_{22}\mathbf{I}\right)\tilde{\mathbf{x}}_{k+1} = \mathbf{b}_2 - r_{12}\tilde{\mathbf{x}}_k. \tag{2.11}$$

Combining (2.10) and (2.11) gives

$$\{(\mathbf{R}_1 + r_{22}\mathbf{I})(\mathbf{R}_1 + r_{11}\mathbf{I}) - r_{12}r_{21}\mathbf{I}\}\tilde{\mathbf{x}}_k = (\mathbf{R}_1 + r_{22}\mathbf{I})\mathbf{b}_1 - r_{21}\mathbf{b}_2. \tag{2.12}$$

Solving (2.12) we get $\tilde{\mathbf{x}}_k$. We solve (2.12) using the following equivalent equation:

$$\{\mathbf{R}_1^2 + (r_{11} + r_{22})\mathbf{R}_1 + (r_{11}r_{22} - r_{12}r_{21})\mathbf{I}\}\tilde{\mathbf{x}}_k = \mathbf{R}_1\mathbf{b}_1 + r_{22}\mathbf{b}_1 - r_{21}\mathbf{b}_2. \tag{2.13}$$

This avoids computing $(\mathbf{R}_1 + r_{22}\mathbf{I})(\mathbf{R}_1 + r_{11}\mathbf{I})$ at each step, we compute $\mathbf{R}_1^2$ only once at the beginning and use it throughout. Solving (2.13) for $\tilde{\mathbf{x}}_k$ provides one scheme for $\tilde{\mathbf{x}}_{k+1}$, namely, the *1-solve scheme*: plug $\tilde{\mathbf{x}}_k$ into (2.10) to get $\tilde{\mathbf{x}}_{k+1}$, that is, we obtain $\{\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}\}$ by solving only one (quasi)-triangular system. Another straightforward way to obtain $\tilde{\mathbf{x}}_{k+1}$ is to plug $\tilde{\mathbf{x}}_k$ into (2.11), then solve for $\tilde{\mathbf{x}}_{k+1}$.

We can do much better by rearranging (2.9),

$$\tilde{\mathbf{x}}_k = \frac{1}{r_{12}}\{\mathbf{b}_2 - (\mathbf{R}_1 + r_{22}\mathbf{I})\tilde{\mathbf{x}}_{k+1}\}. \tag{2.14}$$

Combining (2.14) and (2.8) as above, we get

$$\{\mathbf{R}_1^2 + (r_{11} + r_{22})\mathbf{R}_1 + (r_{11}r_{22} - r_{12}r_{21})\mathbf{I}\}\tilde{\mathbf{x}}_{k+1} = \mathbf{R}_1\mathbf{b}_2 + r_{11}\mathbf{b}_2 - r_{12}\mathbf{b}_1. \tag{2.15}$$

From (2.13) and (2.15), we see that $\{\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}\}$ now can be solved simultaneously by solving the following equation:

$$\{\mathbf{R}_1^2 + (r_{11} + r_{22})\mathbf{R}_1 + (r_{11}r_{22} - r_{12}r_{21})\mathbf{I}\}[\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}] = [\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2], \tag{2.16}$$

where

$$\begin{aligned} \tilde{\mathbf{b}}_1 &= \mathbf{R}_1\mathbf{b}_1 + r_{22}\mathbf{b}_1 - r_{21}\mathbf{b}_2, \\ \tilde{\mathbf{b}}_2 &= \mathbf{R}_1\mathbf{b}_2 + r_{11}\mathbf{b}_2 - r_{12}\mathbf{b}_1. \end{aligned} \tag{2.17}$$

We call this scheme the *2-solve scheme*. This scheme is attractive because (2.16) can be solved by calling block version system-solve routine in LAPACK [1], which is usually more efficient than solving for columns sequentially.

Our third scheme is also aimed at applying block version (quasi-)triangular system-solve routine to solve for $\{\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}\}$ simultaneously. It is very similar to the above 2-solve scheme but derived in a different way.

We denote this scheme as *E-solve scheme* since it uses eigendecomposition. The derivation is as follows: from (2.7) we get

$$\mathbf{R}_1\left[\tilde{\mathbf{x}}_k,\tilde{\mathbf{x}}_{k+1}\right]+\left[\tilde{\mathbf{x}}_k,\tilde{\mathbf{x}}_{k+1}\right]\begin{bmatrix} r_{11} & r_{12}\\ r_{21} & r_{22}\end{bmatrix}=\left[\mathbf{b}_1,\mathbf{b}_2\right]. \tag{2.18}$$

Let the real eigendecomposition of $\left[\begin{smallmatrix} r_{11} & r_{12}\\ r_{21} & r_{22}\end{smallmatrix}\right]$ be

$$\begin{bmatrix} r_{11} & r_{12}\\ r_{21} & r_{22}\end{bmatrix}=\left[\mathbf{y}_1,\mathbf{y}_2\right]\begin{bmatrix} \mu & \eta\\ -\eta & \mu\end{bmatrix}\left[\mathbf{y}_1,\mathbf{y}_2\right]^{-1}, \tag{2.19}$$

where $\mathbf{y}_1,\mathbf{y}_2\in\mathbb{R}^2$. Then (2.18) is equivalent to

$$\mathbf{R}_1\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]+\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]\begin{bmatrix} \mu & \eta\\ -\eta & \mu\end{bmatrix}=\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right], \tag{2.20}$$

where

$$\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]=\left[\tilde{\mathbf{x}}_k,\tilde{\mathbf{x}}_{k+1}\right]\left[\mathbf{y}_1,\mathbf{y}_2\right],\qquad \left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right]=\left[\mathbf{b}_1,\mathbf{b}_2\right]\left[\mathbf{y}_1,\mathbf{y}_2\right]. \tag{2.21}$$

From (2.20), we get

$$\left(\mathbf{R}_1+\mu\mathbf{I}\right)\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]+\eta\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]\begin{bmatrix} 0 & 1\\ -1 & 0\end{bmatrix}=\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right], \tag{2.22}$$

and multiplying $(\mathbf{R}_1+\mu\mathbf{I})$ on both sides of (2.22) we get

$$\left(\mathbf{R}_1+\mu\mathbf{I}\right)^2\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]+\eta\left(\mathbf{R}_1+\mu\mathbf{I}\right)\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]\begin{bmatrix} 0 & 1\\ -1 & 0\end{bmatrix}=\left(\mathbf{R}_1+\mu\mathbf{I}\right)\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right]. \tag{2.23}$$

Combining (2.22) and (2.23) and noting that $\left[\begin{smallmatrix} 0 & 1\\ -1 & 0\end{smallmatrix}\right]^2=\left[\begin{smallmatrix} -1 & 0\\ 0 & -1\end{smallmatrix}\right]$, we finally get

$$\begin{aligned}\left(\left(\mathbf{R}_1+\mu\mathbf{I}\right)^2+\eta^2\mathbf{I}\right)&\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]\\ &=-\eta\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right]\begin{bmatrix} 0 & 1\\ -1 & 0\end{bmatrix}+\left(\mathbf{R}_1+\mu\mathbf{I}\right)\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right]\end{aligned} \tag{2.24}$$

or, equivalently,

$$\left(\mathbf{R}_1^2+2\mu\mathbf{R}_1+(\mu^2+\eta^2)\mathbf{I}\right)\left[\hat{\mathbf{x}}_k,\hat{\mathbf{x}}_{k+1}\right]=\eta\left[\hat{\mathbf{b}}_2,-\hat{\mathbf{b}}_1\right]+\left(\mathbf{R}_1+\mu\mathbf{I}\right)\left[\hat{\mathbf{b}}_1,\hat{\mathbf{b}}_2\right]. \tag{2.25}$$

From (2.25), we can solve $\{\hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k+1}\}$ simultaneously by calling block version system-solve routine in LAPACK. Finally,

$$[\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_{k+1}] = [\hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k+1}] [\mathbf{y}_1, \mathbf{y}_2]^{-1} \tag{2.26}$$

leads to the two columns we need for system (2.18).

Remark 2.1. Even though the 2-solve and E-solve schemes are derived in two different ways, they are closely related to each other because of the following relation of each 2×2 block:

$$2\mu = r_{11} + r_{22}, \qquad \mu^2 + \eta^2 = r_{11}r_{22} - r_{12}r_{21}. \tag{2.27}$$

Hence, the left-hand side coefficient matrices in (2.16) and (2.25) are theoretically exactly the same, while 2-solve does not need to perform eigendecomposition of each 2×2 block and the back transformation (2.26), hence 2-solve is a bit more efficient and accurate than E-solve. However, the derivation of E-solve is in itself interesting and would be useful in developing block algorithms.

In the next sections, we will mainly use the 2-solve scheme since it is the most accurate and efficient among the three schemes.

2.2. $\mathbf{A}_2$ *is transformed into lower quasitriangular matrix and other cases*

We address the case when $\mathbf{A}_2$ is orthogonally transformed into lower quasitriangular matrix. This case is not essential since even for Lyapunov equation $\mathbf{AX} + \mathbf{XA}^T + \mathbf{D} = \mathbf{0}$, we can apply Algorithm 2.1 without using "one upper triangular-one lower triangular" form by a simple reordering. But the direct "one upper triangular-one lower triangular" form is more natural for Lyapunov equations, hence we briefly mention the difference in the formula for the sake of completeness.

Let $\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1^T$ as before, let $\mathbf{A}_2 = \mathbf{Q}_2\mathbf{L}_2\mathbf{Q}_2^T$, and let $\mathbf{L}_2$ be lower quasitriangular.

Denote $\mathbf{L}_2 = [l_{ij}]$, $i, j = 1, \dots, n$. The transformed equation corresponding to (1.1) is

$$\mathbf{R}_1\tilde{\mathbf{X}} + \tilde{\mathbf{X}}\mathbf{L}_2 + \tilde{\mathbf{D}} = \mathbf{0}, \quad \text{where } \tilde{\mathbf{D}} = \mathbf{Q}_1^T\mathbf{D}\mathbf{Q}_2. \tag{2.28}$$

Denote $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \dots, \tilde{\mathbf{x}}_n]$ and $\tilde{\mathbf{D}} = [\tilde{\mathbf{d}}_1, \tilde{\mathbf{d}}_2, \dots, \tilde{\mathbf{d}}_n]$, where $\tilde{\mathbf{x}}_i, \tilde{\mathbf{d}}_i \in \mathbb{R}^m$. Now, we need to solve $\tilde{\mathbf{X}}$ by backward substitutions.

For $k = m$ down to $k = 1$, when $l_{k-1,k} = 0$, we solve for $\tilde{\mathbf{x}}_k$ by

$$\left(\mathbf{R}_1 + l_{k,k}\mathbf{I}\right)\tilde{\mathbf{x}}_k = -\tilde{\mathbf{d}}_k - \sum_{i=k+1}^{n} \tilde{\mathbf{x}}_i l_{i,k}. \tag{2.29}$$

When $l_{k-1,k} \neq 0$, we solve for $\{\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{x}}_k\}$ simultaneously. Since at this kth step

$$\begin{aligned} \mathbf{R}_1\left[\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{x}}_k\right] + \left[\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{x}}_k\right]\begin{bmatrix} l_{k-1,k-1} & l_{k-1,k} \\ l_{k,k-1} & l_{k,k} \end{bmatrix} \\ = -\left[\tilde{\mathbf{d}}_{k-1}, \tilde{\mathbf{d}}_k\right] - \left[\sum_{i=k+1}^{n} \tilde{\mathbf{x}}_i l_{i,k-1}, \sum_{i=k+1}^{n} \tilde{\mathbf{x}}_i l_{i,k}\right], \end{aligned} \tag{2.30}$$

the two vectors in the right-hand side are known. Denote them as $[\mathbf{b}_1, \mathbf{b}_2]$, then the same column-wise elimination scheme in Section 2.1 can be applied which leads to the following equation:

$$\left\{\mathbf{R}_1^2 + \left(l_{k-1,k-1} + l_{k,k}\right)\mathbf{R}_1 + \left(l_{k-1,k-1}l_{k,k} - l_{k-1,k}l_{k,k-1}\right)\mathbf{I}\right\}\left[\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{x}}_k\right] = \left[\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2\right], \tag{2.31}$$

where

$$\begin{aligned} \tilde{\mathbf{b}}_1 &= \mathbf{R}_1\mathbf{b}_1 + l_{k,k}\mathbf{b}_1 - l_{k,k-1}\mathbf{b}_2, \\ \tilde{\mathbf{b}}_2 &= \mathbf{R}_1\mathbf{b}_2 + l_{k-1,k-1}\mathbf{b}_2 - l_{k-1,k}\mathbf{b}_1. \end{aligned} \tag{2.32}$$

Solving (2.31), we get $\{\tilde{\mathbf{x}}_{k-1}, \tilde{\mathbf{x}}_k\}$, and the process can be carried on until $k = 1$.

Note that for Sylvester equation of form (1.1), if we solve $\mathbf{X}$ column by column, then we only need to look at the diagonal blocks of the transformed quasitriangular matrix of $\mathbf{A}_2$.

We can also solve $\mathbf{X}$ row by row (i.e., by taking the transpose of (1.1), then solving column by column). In this case, we only need to look at the diagonal blocks of the transformed quasitriangular matrix of $\mathbf{A}_1^T$, the choice of this approach is necessary when $\mathbf{R}_1$ is much more ill-conditioned than $\mathbf{R}_2$ or $\mathbf{L}_2$. The solution formula for all these cases may be derived essentially in the same way as what have been discussed in Sections 2.1 and 2.2.

2.3. Computational complexity

We note that our schemes can be adapted easily to use the Hessenberg-Schur method [9]; the only modification necessary is in the first step:

instead of doing Schur decomposition of $\mathbf{A}_1$ we do Hessenberg decomposition $\mathbf{A}_1 = \mathbf{Q}_1\mathbf{H}_1\mathbf{Q}_1^T$, where $\mathbf{H}_1$ is upper Hessenberg. This saves flops when $\mathbf{A}_1$ and $\mathbf{A}_2$ have no connection (i.e., $\mathbf{A}_1 \neq \mathbf{A}_2$ or $\mathbf{A}_2^T$, the Schur decomposition of $\mathbf{A}_2$ provides no information for the Schur decomposition of $\mathbf{A}_1$). The numerical study we will present in Section 2.4 still uses the Schur decomposition because we also deal with the Lyapunov equation and the case $\mathbf{A}_1 = \mathbf{A}_2$. For these two cases, the Hessenberg-Schur method offers no advantage.

The computational advantage of our 2-solve scheme can be seen from the following perspective. It is clear that for the real eigenvalues of $\mathbf{A}_2$, our scheme is the same as Bartels-Stewart algorithm (or Hessenberg-Schur algorithm if we apply the above-mentioned modification). The major difference is in how to deal with the complex eigenvalues of $\mathbf{A}_2$. All the other flop counts in [3, 9] work the same for our scheme.

As seen from (2.7) or (2.30), the Bartels-Stewart and Hessenberg-Schur algorithms essentially solve the corresponding $(2m \times 2m)$-linear equation equivalent to (2.7) or (2.30) (via Kronecker product). With a suitable reordering, the coefficient matrix becomes upper triangular with two nonzero subdiagonals in the lower triangular part. The flop count for this $(2m \times 2m)$-equation, without counting the formation of the right-hand side vector, is $6m^2$ as counted in [9]. Additional $2m^2$ storage is required.

While we solve the equivalent $(m \times m)$-order equation (2.16) or (2.31), the coefficient matrix is quasi-upper triangular, hence the solution flop is m^2 (each column costs $m^2/2$ flops). Updating the two right-hand side vectors via the additional multiplication by $\mathbf{R}_1$ costs about m^2 flops (one triangular matrix vector product costs about $m^2/2$). Forming $\mathbf{R}_1^2$ once costs about $m^3/6$, this adds about $m^3/6n_0$ flops for each two columns of the solution, where n_0 is the number of conjugate eigenpairs of $\mathbf{A}_2$. Forming the coefficient matrix in (2.16) or (2.31) costs about $m^2/2$ flops. So the total flop is about $5m^2/2 + m^3/6n_0$ for each two columns of the solution. We see that the more conjugate eigenpairs $\mathbf{A}_2$ has the fewer flops our scheme requires in comparison to Bartels-Stewart or Hessenberg-Schur algorithm. We also note that when n_0 is small (i.e., $n_0 < m/21$), then forming $\mathbf{R}_1^2$ may not be worthwhile; in this case we would use standard Bartels-Stewart or Hessenberg-Schur algorithm. In the case $n_0 > m/21$, our scheme uses less flop than Bartels-Stewart or Hessenberg-Schur algorithm.

Certainly, the flop count is not the only issue in determining the efficiency of an algorithm. We point out that the major advantage of our column-wise elimination scheme (which avoids equivalent Kronecker form) is that we can call block version system-solve routines in LAPACK [1], no reordering or additional data movement is necessary. This feature is more suitable for many computer architectures where cache performance is an important issue. Also, the Kronecker form implies $(2m)^2$ additional storage; by suitable reordering, $2m^2$ additional storage is usually necessary. While we need m^2 additional storage for $\mathbf{R}_1^2$.

The disadvantage of our scheme is in conditioning. If $\mathbf{R}_1$ is ill-conditioned, forming $\mathbf{R}_1^2$ can be numerically problematic.

2.4. Algorithm and numerical results

We present the algorithm of the 2-solve scheme for Sylvester equations in Algorithm 2.1. The 1-solve and E-solve schemes can be coded by small modifications on the 2-solve scheme. In the algorithm, we handled the case that $\mathbf{A}_2$ is transformed into quasi-upper triangular matrix. For other cases discussed in Section 2.2, the codes may be developed similarly.

Note that our algorithm also handle the Lyapunov equation

$$\mathbf{AX} + \mathbf{XA}^T + \mathbf{D} = \mathbf{0}, \tag{2.33}$$

and the special Sylvester equation of the form

$$\mathbf{AX} + \mathbf{XA} + \mathbf{D} = \mathbf{0}. \tag{2.34}$$

We point out that the current Matlab code *lyap.m* does not solve (2.34) as efficiently as possible since it performs two complex Schur decomposition to the same matrix $\mathbf{A}$.

We report some computational comparison between our schemes and the Matlab function *lyap.m*, the computation is done in Matlab 6.0 on a Dell Dimension L800r PC (Intel Pentium III, 800 MHz CPU, 128 MB memory) with operating system Win2000. Figure 2.1 shows the comparison between *lyap.m* and our three schemes for the special Sylvester equation (2.34). Figure 2.2 is about the Lyapunov equation (2.33). These results show that our schemes are much more efficient than the Matlab *lyap.m*, the CPU time difference between *lyap.m* and our schemes will be greater when n becomes larger. And the accuracy of our schemes is similar to that of *lyap.m*.

These results also show that even though the 1-solve scheme theoretically use less flops than the 2-solve scheme, it does not necessary use less CPU time. In modern computer architecture, the cache performance and data communication speed are also important aspects [6]. The 2-solve scheme often may have better cache performance than the 1-solve scheme (for the former, matrix $\mathbf{A}$ needs to be accessed only once for two system solves, this reduces memory traffic), hence it is able to consume less CPU time when n becomes larger. This observation also supports our choice in not using the Kronecker form.

```
Input data: $\mathbf{A}_1 \in \mathbb{R}^{m\times m}$, $\mathbf{A}_2 \in \mathbb{R}^{n\times n}$, $\mathbf{D} \in \mathbb{R}^{m\times n}$
Output data: solution $\mathbf{X} \in \mathbb{R}^{m\times n}$

    (1) Compute real Schur decomposition of $\mathbf{A}_1$, $\mathbf{A}_1 = \mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1^T$.
    (2) If $\mathbf{A}_2 = \mathbf{A}_1$, set $\mathbf{Q}_2 = \mathbf{Q}_1$, $\mathbf{R}_2 = \mathbf{R}_1$;

        else if $\mathbf{A}_2 = \mathbf{A}_1^T$, get $\mathbf{Q}_2$, $\mathbf{R}_2$ from $\mathbf{Q}_1$, $\mathbf{R}_1$ as follows:
        $idx = [\text{size}(\mathbf{A}_1,1) : -1 : 1]$; $\mathbf{Q}_2 = \mathbf{Q}_1(:,idx)$;
        $\mathbf{R}_2 = \mathbf{R}_1(idx,idx)^T$;

        else, compute real Schur decomposition of $\mathbf{A}_2$, $\mathbf{A}_2 = \mathbf{Q}_2\mathbf{R}_2\mathbf{Q}_2^T$.
    (3) $\mathbf{D} \leftarrow \mathbf{Q}_1^T\mathbf{D}\mathbf{Q}_2$, $Rsq \leftarrow \mathbf{R}_1 * \mathbf{R}_1$, $\mathbf{I} \leftarrow eye(m)$, $j \leftarrow 1$.
    (4) While $(j < n+1)$

        if $j < n$ and $\mathbf{R}_2(j+1,j) < 10 * \epsilon * \max(|\mathbf{R}_2(j,j)|,|\mathbf{R}_2(j+1,j+1)|)$
          (a) $\mathbf{b} \leftarrow -\mathbf{D}(:,j) - \mathbf{X}(:,1:j-1) * \mathbf{R}_2(1:j-1,j)$;
          (b) solve the linear equations $(\mathbf{R}_1 + \mathbf{R}_2(j,j)\mathbf{I})\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, set $\mathbf{X}(:,j) \leftarrow \mathbf{x}$;
          (c) $j \leftarrow j+1$
        else
          (a) $r_{11} \leftarrow \mathbf{R}_2(j,j)$, $r_{12} \leftarrow \mathbf{R}_2(j,j+1)$,
              $r_{21} \leftarrow \mathbf{R}_2(j+1,j)$, $r_{22} \leftarrow \mathbf{R}_2(j+1,j+1)$;
          (b) $\mathbf{b} \leftarrow -\mathbf{D}(:,j:j+1) - \mathbf{X}(:,1:j-1) * \mathbf{R}_2(1:j-1,j:j+1)$;
              $\mathbf{b} \leftarrow [\mathbf{R}_1\mathbf{b}(:,1) + r_{22}\mathbf{b}(:,1) - r_{21}\mathbf{b}(:,2), \mathbf{R}_1\mathbf{b}(:,2) + r_{11}\mathbf{b}(:,2) - r_{12}\mathbf{b}(:,1)]$
          (c) block solve the linear equations
              $(Rsq + (r_{11}+r_{22})\mathbf{R}_1 + (r_{11}r_{22} - r_{12}r_{21})\mathbf{I})\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, set $\mathbf{X}(:,j:j+1) \leftarrow \mathbf{x}$;
          (d) $j \leftarrow j+2$
        end if.
    (5) The solution $\mathbf{X}$ in the original basis is: $\mathbf{X} \leftarrow \mathbf{Q}_1\mathbf{X}\mathbf{Q}_2^T$.
```

ALGORITHM 2.1. The 2-solve scheme for Sylvester equation $\mathbf{A}_1\mathbf{X} + \mathbf{X}\mathbf{A}_2 + \mathbf{D} = \mathbf{0}$.

The accuracy of the 1-solve scheme is not as high as the 2-solve scheme (this can be explained by (2.10), the errors in $\tilde{\mathbf{x}}_k$ would be magnified if r_{21} is small). The 2-solve scheme is much better since no explicit divides are performed.

3. Hammarling's method and new formulations

Since the solution $\mathbf{X}$ of (1.2) is at least semidefinite, Hammarling found an ingenuous way to compute the Cholesky factor of $\mathbf{X}$ directly. His paper [11] remains the main (and the only) reference in this direction since its first appearance in 1982. Penzl in [16] generalized exactly the same technique to the generalized Lyapunov equation.

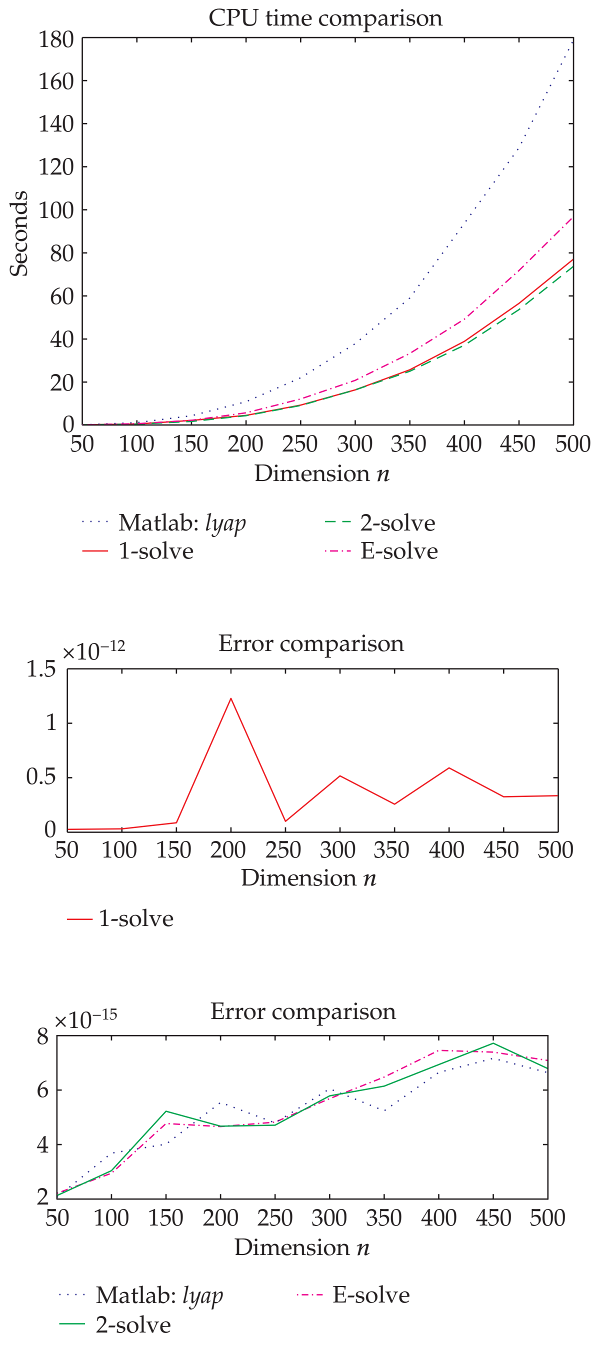


FIGURE 2.1. Comparison of performance for Sylvester equation (2.34).

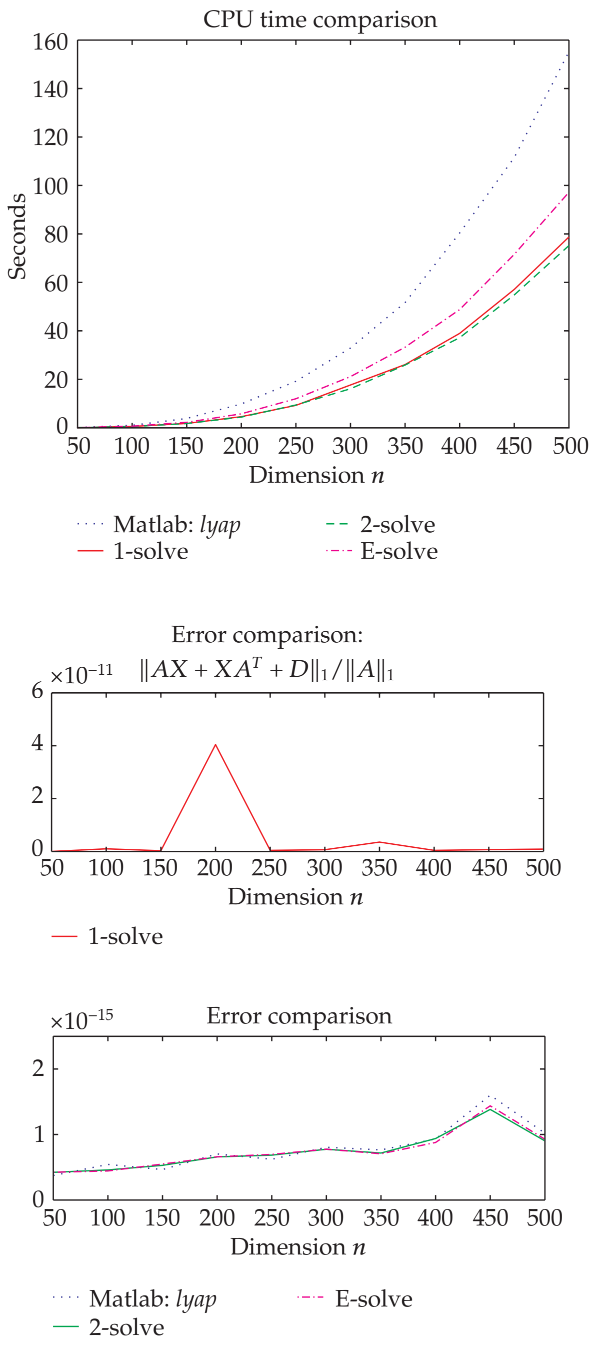


FIGURE 2.2. Comparison of performance for Lyapunov equation (2.33).

3.1. Solving by complex arithmetic

We first discuss the method using complex arithmetic. Hammarling's method depends on the following observation: triangular structure naturally allows forward or backward substitution, that is, there is an intrinsic recursive-structure algorithm for triangular systems. Algorithms for triangular matrix equations in a recursive style are derived in Jonsson and Kågström [14]. The original paper of the Bartels-Stewart algorithm [3] used orthogonal transformation to transform one coefficient matrix $\mathbf{A}_2$ to upper triangular form and the other matrix $\mathbf{A}_1$ to lower triangular form, then the transformed triangular matrix equation can be reduced to another triangular matrix equation with the same structure but with 1 or 2 orders less. Hammarling [11] explained this reduction very clearly using Lyapunov equation of the form

$$\mathbf{A}^H\mathbf{X}+\mathbf{X}\mathbf{A}+\mathbf{D}=\mathbf{0} \tag{3.1}$$

as an example, and he used backward substitution.

Here, we consider Lyapunov equation of the form

$$\mathbf{A}\mathbf{X}+\mathbf{X}\mathbf{A}^H+\mathbf{D}=\mathbf{0}, \quad \text{where } \mathbf{D}=\mathbf{D}^H. \tag{3.2}$$

We point out that essentially the same reduction process also works for Sylvester equations.

Let the Schur decomposition of $\mathbf{A}$ be

$$\mathbf{A}=\mathbf{Q}\mathbf{R}\mathbf{Q}^H, \tag{3.3}$$

let $\tilde{\mathbf{X}}=\mathbf{Q}^H\mathbf{X}\mathbf{Q}$ and $\tilde{\mathbf{D}}=\mathbf{Q}^H\mathbf{D}\mathbf{Q}$, then (3.2) becomes

$$\mathbf{R}\tilde{\mathbf{X}}+\tilde{\mathbf{X}}\mathbf{R}^H+\tilde{\mathbf{D}}=\mathbf{0}. \tag{3.4}$$

Partition $\mathbf{R}$, $\tilde{\mathbf{X}}$, and $\tilde{\mathbf{D}}$ as follows:

$$\begin{aligned}\mathbf{R}&=\begin{bmatrix}\mathbf{R}_1 & \mathbf{r}\\ \mathbf{0} & \lambda_n\end{bmatrix},\\ \tilde{\mathbf{X}}&=\begin{bmatrix}\mathbf{X}_1 & \mathbf{x}\\ \mathbf{x}^H & x_{nn}\end{bmatrix},\\ \tilde{\mathbf{D}}&=\begin{bmatrix}\mathbf{D}_1 & \mathbf{d}\\ \mathbf{d}^H & d_{nn}\end{bmatrix},\end{aligned} \tag{3.5}$$

where $\mathbf{R}_1, \mathbf{X}_1, \mathbf{D}_1 \in \mathbb{C}^{(n-1)\times(n-1)}$ and $\mathbf{r}, \mathbf{x}, \mathbf{d} \in \mathbb{C}^{n-1}$. Then (3.4) gives three equations

$$(\lambda_n + \bar{\lambda}_n) x_{nn} + d_{nn} = 0, \tag{3.6}$$

$$(\mathbf{R}_1 + \bar{\lambda}_n \mathbf{I})\mathbf{x} + \mathbf{d} + x_{nn}\mathbf{r} = \mathbf{0}, \tag{3.7}$$

$$\mathbf{R}_1\mathbf{X}_1 + \mathbf{X}_1\mathbf{R}_1^H + \mathbf{D}_1 + \mathbf{r}\mathbf{x}^H + \mathbf{x}\mathbf{r}^H = \mathbf{0}. \tag{3.8}$$

From (3.6), we get

$$x_{nn} = -\frac{d_{nn}}{(\lambda_n + \bar{\lambda}_n)}. \tag{3.9}$$

Plugging x_{nn} in (3.7), we can solve for $\mathbf{x}$, after $\mathbf{x}$ is known, and (3.8) becomes a Lyapunov equation which has the same structure as (3.4) but of order $(n-1)$:

$$\mathbf{R}_1\mathbf{X}_1 + \mathbf{X}_1\mathbf{R}_1^H = -\mathbf{D}_1 - \mathbf{r}\mathbf{x}^H - \mathbf{x}\mathbf{r}^H. \tag{3.10}$$

We can apply the same process to (3.10) till $\mathbf{R}_1$ is of order 1. Note under the condition that $\lambda_i + \bar{\lambda}_i \neq 0$, $i = 1,\dots,n$, at the kth step $(k = 1,2,\dots,n)$ of this process, we get a unique solution vector of length $(n+1-k)$ and a reduced triangular matrix equation of order $(n-k)$.

Hammarling's idea [11] was based on this recursive reduction (3.10). He observed that when $\mathbf{D} = \mathbf{B}\mathbf{B}^T$ in (3.2), it is possible to compute the Cholesky factor of $\mathbf{X}$ directly without forming $\mathbf{D}$. The difficulty in the reduction process includes expressing the Cholesky factor of the right-hand side of (3.10) as a rank-1 update to the Cholesky factor of $\mathbf{D}_1$ without forming $\mathbf{D}_1$. For more details, see [11].

In [11], the case where $\mathbf{B}$ has more columns than rows is emphasized. When $\mathbf{B}$ has more rows than columns, a different updating scheme must be used. Our updating scheme is able to treat both of these cases. In LTI system model reduction, we are primarily interested in $\mathbf{B} \in \mathbb{R}^{n\times p}$, where $p \ll n$ and (A, B) is controllable [19]. Our scheme naturally includes this case. We mainly use block Gauss elimination, and our derivation in computing the Cholesky factor directly is perhaps more systematic than the derivation in [11].

We first discuss the complex case. We will treat the transformed triangular Lyapunov equation

$$\mathbf{R}\mathbf{P} + \mathbf{P}\mathbf{R}^H + \mathbf{B}\mathbf{B}^H = \mathbf{0}, \tag{3.11}$$

where $\mathbf{R}$ comes from (3.3), $\mathbf{R}$ is stable, $\mathbf{P} \leftarrow \mathbf{Q}^H\mathbf{P}\mathbf{Q}$, and $\mathbf{B} \leftarrow \mathbf{Q}^H\mathbf{B}$.

Let $\mathbf{P} = \mathbf{U}\mathbf{U}^H$ be the Cholesky decomposition of $\mathbf{P}$; we want to compute $\mathbf{U}$ directly. Partition $\mathbf{U}$ as

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{u} \\ & \tau \end{bmatrix}, \tag{3.12}$$

where $\mathbf{U}_1 \in \mathbb{C}^{(n-1)\times(n-1)}$, $\mathbf{u} \in \mathbb{C}^{n-1}$, and $\tau \in \mathbb{C}$. Under the controllability assumption, $\mathbf{P} > 0$, so we set $\tau > 0$. Then

$$\mathbf{P} = \mathbf{U}\mathbf{U}^H = \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^H + \mathbf{u}\mathbf{u}^H & \tau\mathbf{u} \\ \tau\mathbf{u}^H & \tau^2 \end{bmatrix}. \tag{3.13}$$

The main idea is to block-diagonalize $\mathbf{P}$, this can be done via an elementary matrix $\left[\begin{smallmatrix} \mathbf{I} & -(1/\tau)\mathbf{u} \\ & 1 \end{smallmatrix}\right]$ since it can be verified that

$$\begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \mathbf{P} \begin{bmatrix} \mathbf{I} & \\ -\frac{1}{\tau}\mathbf{u}^H & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^H & \\ & \tau^2 \end{bmatrix}. \tag{3.14}$$

By some well-known inverse formula of elementary matrices, we get

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} & \frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^H & \\ & \tau^2 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ \frac{1}{\tau}\mathbf{u}^H & 1 \end{bmatrix}. \tag{3.15}$$

Plugging (3.15) into (3.11) and multiplying two elementary matrices from the left and from the right, we get

$$\begin{aligned} &\left(\begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \mathbf{R} \begin{bmatrix} \mathbf{I} & \frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \right) \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^H & \\ & \tau^2 \end{bmatrix} \\ &\quad + \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^H & \\ & \tau^2 \end{bmatrix} \left(\begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} R \begin{bmatrix} \mathbf{I} & \frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \right)^H \\ &\quad + \begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} BB^H \begin{bmatrix} \mathbf{I} & \\ -\frac{1}{\tau}\mathbf{u}^H & 1 \end{bmatrix} = 0. \end{aligned} \tag{3.16}$$

Partition **B** and **R** as

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{b}^H \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{r} \\ & \lambda \end{bmatrix}, \tag{3.17}$$

where **r**, **b** are vectors and λ is a scalar. We get

$$\begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 - \frac{1}{\tau}\mathbf{u}\mathbf{b}^H \\ \mathbf{b}^H \end{bmatrix} := \begin{bmatrix} \hat{\mathbf{B}}_1 \\ \mathbf{b}^H \end{bmatrix}, \tag{3.18}$$

where the rank-1 update of $\mathbf{B}_1$ is

$$\hat{\mathbf{B}}_1 = \mathbf{B}_1 - \frac{1}{\tau}\mathbf{u}\mathbf{b}^H, \tag{3.19}$$

$$\begin{bmatrix} \mathbf{I} & -\frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} \mathbf{R} \begin{bmatrix} \mathbf{I} & \frac{1}{\tau}\mathbf{u} \\ & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1 & \frac{1}{\tau}(\mathbf{R}_1 - \lambda I)\mathbf{u} + \mathbf{r} \\ & \lambda \end{bmatrix}. \tag{3.20}$$

Plugging (3.18) and (3.20) into (3.16), we get

$$\begin{bmatrix} \mathbf{R}_1(\mathbf{U}_1\mathbf{U}_1^H) & \left(\frac{1}{\tau}(\mathbf{R}_1 - \lambda I)\mathbf{u} + \mathbf{r}\right)\tau^2 \\ & \lambda\tau^2 \end{bmatrix} + \begin{bmatrix} (\mathbf{U}_1\mathbf{U}_1^H)\mathbf{R}_1^H & \\ \left(\frac{1}{\tau}(\mathbf{R}_1 - \lambda \mathbf{I})\mathbf{u} + \mathbf{r}\right)^H \tau^2 & \bar{\lambda}\tau^2 \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{B}}_1\hat{\mathbf{B}}_1^H & \hat{\mathbf{B}}_1\mathbf{b} \\ (\hat{\mathbf{B}}_1\mathbf{b})^H & \mathbf{b}^H\mathbf{b} \end{bmatrix} = 0. \tag{3.21}$$

Equation (3.21) contains exactly the same recursive structure that we have just discussed in (3.4), (3.6), (3.7), and (3.8), that is, (3.21) leads to three equations

$$(\lambda + \bar{\lambda})\tau^2 + \mathbf{b}^H\mathbf{b} = 0, \tag{3.22}$$

$$(\mathbf{R}_1 - \lambda I)\mathbf{u} + \tau\mathbf{r} + \frac{1}{\tau}\hat{\mathbf{B}}_1\mathbf{b} = \mathbf{0}, \tag{3.23}$$

$$\mathbf{R}_1(\mathbf{U}_1\mathbf{U}_1^H) + (\mathbf{U}_1\mathbf{U}_1^H)\mathbf{R}_1^H + \hat{\mathbf{B}}_1\hat{\mathbf{B}}_1^H = \mathbf{0}. \tag{3.24}$$

Under the condition that **R** is stable, (3.22) has a unique solution (since we choose $\tau > 0$)

$$\tau = \frac{\|\mathbf{b}\|_2}{\sqrt{-2\,\mathrm{real}(\lambda)}}. \tag{3.25}$$

```
Input data: $\mathbf{R} \in \mathbb{R}^{n\times n}$, $\mathbf{R}$ is upper triangular and stable, $\mathbf{B} \in \mathbb{R}^{n\times p}$
Output data: the Cholesky factor $\mathbf{U}$ of the solution $\mathbf{P}$, $\mathbf{U} \in \mathbb{R}^{n\times n}$
        $\mathbf{U} \leftarrow n$ by $n$ zero matrix
        for $j = n : -1 : 2$ do
          (1) $\mathbf{b} \leftarrow \mathbf{B}(j,:)$; $\mu \leftarrow \|\mathbf{b}\|_2$, $\mu_1 \leftarrow \sqrt{-2 * \text{real}(\mathbf{R}(j,j))}$
          (2) if $\mu > 0$
              $\mathbf{b} \leftarrow \mathbf{b}/\mu$; $\mathbf{I} \leftarrow (j-1)$ order identity matrix
              $btmp \leftarrow \mathbf{B}(1:j-1,:) * \mathbf{b}^H * \mu_1 + \mathbf{R}(1:j-1,j) * \mu/\mu_1$
              solve $(\mathbf{R}(1:j-1,1:j-1)+\mathbf{R}(j,j)^H * \mathbf{I})\mathbf{u} = -btmp$ for $\mathbf{u}$,
                  $\mathbf{B}(1:j-1,:) \leftarrow \mathbf{B}(1:j-1,:) - \mathbf{u} * \mathbf{b} * \mu_1$
              else
              $\mathbf{u} \leftarrow$ length $(j-1)$ zero vector
              end if
          (3) $\mathbf{U}(j,j) \leftarrow \mu/\mu_1$
          (4) $\mathbf{U}(1:j-1,j) \leftarrow \mathbf{u}$
        $\mathbf{U}(1,1) \leftarrow \|\mathbf{B}(1,:)\|_2/\sqrt{-2 * \text{real}(\mathbf{R}(1,1))}$
```

ALGORITHM 3.1. Modified Hammarling's algorithm using complex arithmetic.

Plugging (3.25) and (3.19) into (3.23), we get the formula for $\mathbf{u}$

$$(\mathbf{R}_1 + \bar{\lambda}\mathbf{I})\mathbf{u} = -\tau\mathbf{r} - \frac{1}{\tau}\mathbf{B}_1\mathbf{b}. \tag{3.26}$$

Solving (3.26) gives $\mathbf{u}$, so we get the last column of $\mathbf{U}$. Plugging τ and $\mathbf{u}$ into (3.19), we see that (3.24) is now an order $(n-1)$ triangular Lyapunov equation, with $\mathbf{U}_1$ the only unknown. We can solve (3.24) using the same technique: first solve the last column of $\mathbf{U}_1$, then reduce it to another triangular Lyapunov of order $(n-2)$. This process can be carried on till $\mathbf{R}_1$ is of order 1, then all the unknown elements of $\mathbf{U}$ are solved. One key step at each stage of the process is the rank-1 update formula (3.19) that we got via the elementary matrix manipulations.

The above derivation is summarized in Algorithm 3.1.

Besides the less flop counts, another main advantage in computing Cholesky factor $\mathbf{U}$ directly instead of the full solution $\mathbf{P}$ is that $\kappa(\mathbf{P}) = \kappa(\mathbf{U})^2$. When the Lyapunov equation (3.11) is slightly ill-conditioned, that is, the corresponding Kronecker operator $\mathbf{I}\otimes\mathbf{R} + \mathbf{R}\otimes\mathbf{I}$ is not well conditioned, $\kappa(\mathbf{X})$ will be large and the numerical solution of $\mathbf{U}$ will be more accurate than $\mathbf{P}$. A thorough discussion of this conditioning issue is given in [11]. We will not discuss conditioning in this section. However, we do wish to point out that when the original Lyapunov equation is highly ill-conditioned, the direct implementation of the Bartels-Stewart algorithm or the Hammarling method (even with iterative refinement) will not give a satisfactorily accurate solution. In this case, special effort needs to be taken to handle the ill conditioning. See [8] for one possible approach. A significant motivation for computing Cholesky factor $\mathbf{U}$ directly is from model reduction by balanced truncation. The Hankel singular values can be computed directly from the SVD of the product of the Cholesky factors of the two system Gramians. This avoids an unnecessary squaring that happens if one works directly with the product of Gramians [2, 15].

3.2. Solving by real arithmetic only

Techniques for computing in real arithmetic similar to those developed for the Sylvester equations in Section 2 can be applied to Lyapunov equations with real coefficients. The difficulty again comes from how to handle the (2×2)-blocks in the diagonal of $\mathbf{R}$.

Let the real Schur decomposition of $\mathbf{A}$ be

$$\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{Q}^T, \tag{3.27}$$

where $\mathbf{R} \in \mathbb{R}^{n\times n}$ is real quasi-upper triangular, with diagonal block size less than or equal to 2, a (2×2)-diagonal block corresponds to a conjugate pair of complex eigenvalues.

Again for simplicity of notation we denote the transformed Lyapunov equation as

$$\mathbf{R}\mathbf{P} + \mathbf{P}\mathbf{R}^T + \mathbf{B}\mathbf{B}^T = \mathbf{0}. \tag{3.28}$$

We want to solve for the Cholesky factor $\mathbf{U}$ of $\mathbf{P} = \mathbf{U}\mathbf{U}^T$ directly.

In the case that the diagonal block size is one, we partition $\mathbf{U}$, $\mathbf{R}$, and $\mathbf{B}$ as follows

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{u} \\ & \tau \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{r} \\ & \lambda \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{b}^T \end{bmatrix}, \tag{3.29}$$

where $\mathbf{u}$, $\mathbf{r}$, and $\mathbf{b}$ are vectors and λ, τ are scalars, then

$$\mathbf{P} = \mathbf{U}\mathbf{U}^T = \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^T + \mathbf{u}\mathbf{u}^T & \tau\mathbf{u} \\ \tau\mathbf{u}^T & \tau^2 \end{bmatrix}, \tag{3.30}$$

all the update formulations in Section 3.1 can be applied directly.

Note that the last column of $\mathbf{P}$ is τ times the last column of $\mathbf{U}$. As in the Sylvester equation case, from (3.28), the last column of $\mathbf{P}$ can be solved directly from

$$(\mathbf{R}+\lambda\mathbf{I})x = -\mathbf{B}\mathbf{b}, \quad \text{where } \mathbf{x} = \left(\tau \begin{bmatrix} \mathbf{u} \\ \tau \end{bmatrix}\right). \tag{3.31}$$

Thus from the solution **x** of (3.31), we can obtain τ and **u** as

$$\tau = \sqrt{\mathbf{x}(n)}, \quad \mathbf{u} = \frac{1}{\tau}\mathbf{x}(1:n-1). \tag{3.32}$$

The rank-1 update formula of **B** remains the same as (3.19) except that we only need real arithmetic here,

$$\hat{\mathbf{B}}_1 = \mathbf{B}_1 - \frac{1}{\tau}\mathbf{u}\mathbf{b}^T. \tag{3.33}$$

In the case that the diagonal block of R is of size 2, we partition **U**, R, and B as follows:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{u}_1 & \mathbf{u}_2 \\ & \tau_{11} & \tau_{12} \\ & & \tau_{22} \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{r}_1 & \mathbf{r}_2 \\ & \lambda_{11} & \lambda_{12} \\ & \lambda_{21} & \lambda_{22} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix}, \tag{3.34}$$

where $\mathbf{u}_i$, $\mathbf{r}_i$, and $\mathbf{b}_i$, $i = 1,2$, are vectors and λ_{ij}, τ_{ij}, $i,j = 1,2$, are scalars. Then,

$$\mathbf{P} = \mathbf{U}\mathbf{U}^T = \begin{bmatrix} \mathbf{U}_1\mathbf{U}_1^T + \mathbf{u}_1\mathbf{u}_1^T + \mathbf{u}_2\mathbf{u}_2^T & \tau_{11}\mathbf{u}_1 + \tau_{12}\mathbf{u}_2 & \tau_{22}\mathbf{u}_2 \\ (\text{sym}) & \tau_{11}^2 + \tau_{12}^2 & \tau_{12}\tau_{22} \\ (\text{sym}) & \tau_{12}\tau_{22} & \tau_{22}^2 \end{bmatrix}. \tag{3.35}$$

We also notice that the last two columns of **U** can be obtained from the last two columns of **P**. As discussed in the real Sylvester equation case, from (3.28), the last two columns of **P** can be solved via our column-wise elimination process: the same three 1-solve, 2-solve, and E-solve schemes. Details of the 2-solve scheme can be found from the Matlab codes in [18]. We chose the 2-solve scheme here because it is the best among the three in terms of CPU time and accuracy.

After solving the last two columns of **P** via (3.35), we will be able to get the formula for the last two columns of **U**; the updates of **B** remain the same as (3.33).

The algorithm which fulfills solving (3.28) in only real arithmetic is stated in Algorithm 3.2. The Matlab codes may be found in [18].

```
Input data: $\mathbf{R} \in \mathbb{R}^{n\times n}$, $\mathbf{R}$ is upper triangular and stable, $\mathbf{B} \in \mathbb{R}^{n\times p}$
Output data: the Cholesky factor $\mathbf{U}$ of the solution $\mathbf{P}$, $\mathbf{U} \in \mathbb{R}^{n\times n}$
        set $\mathbf{U} \leftarrow n$ by $n$ zero matrix; $j \leftarrow n$;
        while $(j > 0)$
          if $j > 1$ and $\mathbf{R}(j,j-1) = 0$
                $\mu \leftarrow \mathbf{R}(j,j)$; $\mathbf{b} \leftarrow -\mathbf{B}(1:j,:) * \mathbf{B}(j,:)^H$; $\mathbf{I} \leftarrow eye(j)$;
                solve $(\mathbf{R}(1:j,1:j) + \mu\mathbf{I})\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$; set $\mathbf{U}(1:j,j) \leftarrow \mathbf{x}$,
                if $\mathbf{U}(j,j) > 0$,
                    $\mathbf{U}(j,j) \leftarrow \sqrt{\mathbf{U}(j,j)}$;
                    $\mathbf{U}(1:j-1,j) \leftarrow \mathbf{U}(1:j-1,j)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(j,:) \leftarrow \mathbf{B}(j,:)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(1:j-1,:) \leftarrow \mathbf{B}(1:j-1,:) - \mathbf{U}(1:j-1,j) * \mathbf{B}(j,:)$;
                end if
                $j \leftarrow j-1$
          else
                set $r_{11} \leftarrow \mathbf{R}(j-1,j-1)$; $r_{12} \leftarrow \mathbf{R}(j-1,j)$;
                $r_{21} \leftarrow \mathbf{R}(j,j-1)$; $r_{22} \leftarrow \mathbf{R}(j,j)$;
                $\mathbf{b} \leftarrow -\mathbf{B}(1:j,:) * \mathbf{B}(j-1:j,:)^H$;
                $\mathbf{M} \leftarrow R(1:j,1:j)$; $Msq \leftarrow \mathbf{M} * \mathbf{M}$; $\mathbf{I} \leftarrow eye(j)$;
                $btmp \leftarrow [\mathbf{M} * \mathbf{b}(:,1) + r_{22} * \mathbf{b}(:,1) - r_{21} * \mathbf{b}(:,2),$
                            $\mathbf{M} * \mathbf{b}(:,2) + r_{11} * \mathbf{b}(:,2) - r_{12} * \mathbf{b}(:,1)]$;
                block solve
                $(Msq + (r_{11} + r_{22})\mathbf{M} + (r_{11}r_{22} - r_{12}r_{21})\mathbf{I})\mathbf{x} = btmp$ for $\mathbf{x}$;
                set $\mathbf{U}(1:j,j-1:j) \leftarrow \mathbf{x}$;
                set $\mathbf{U}(j-1,j) \leftarrow (\mathbf{U}(j-1,j) + \mathbf{U}(j,j-1))/2$;
                $\mathbf{U}(j,j-1) \leftarrow 0$;
                if   $\mathbf{U}(j,j) > 0$
                    $\mathbf{U}(j,j) \leftarrow \sqrt{\mathbf{U}(j,j)}$;
                    $\mathbf{U}(1:j-1,j) \leftarrow \mathbf{U}(1:j-1,j)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(j,:) \leftarrow \mathbf{B}(j,:)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(1:j-1,:) \leftarrow \mathbf{B}(1:j-1,:) - \mathbf{U}(1:j-1,j) * \mathbf{B}(j,:)$;
                end if
                $j \leftarrow j-1$
                if   $\mathbf{U}(j,j) > 0$
                    $\mathbf{U}(j,j) \leftarrow \sqrt{\mathbf{U}(j,j)}$;
                    $\mathbf{U}(1:j-1,j) \leftarrow \mathbf{U}(1:j-1,j)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(j,:) \leftarrow \mathbf{B}(j,:)/\mathbf{U}(j,j)$;
                    $\mathbf{B}(1:j-1,:) \leftarrow \mathbf{B}(1:j-1,:) - \mathbf{U}(1:j-1,j) * \mathbf{B}(j,:)$;
                end if
                $j \leftarrow j-1$
          end if
        end while
```

ALGORITHM 3.2. Modified Hammarling's algorithm using real arithmetic only.

3.3. Numerical comparisons and discussions

Figure 3.1 shows the performance comparison of four Lyapunov solvers *lyap.m*, *lyapU.m*, *lyapUR.m*, and *lyapUE.m*, where *lyap.m* is the original function in Matlab, *lyapU.m*, *lyapUR.m*, and *lyapUE.m* are our Lyapunov solvers, *lyapU.m* uses complex arithmetic, *lyapUR.m* uses only real arithmetic with the 2-solve scheme, and *lyapUE.m* uses only real arithmetic with the E-solve scheme. We did not list the 1-solve scheme here since it is not as accurate as the other schemes. Figure 3.1 is one of the many computations we did using Matlab 6.0 in a Dell Dimension L800r PC (Intel Pentium III 800 MHz CPU). Each of the computations obtained similar results. The CPU time shows, as expected, that computing Cholesky factor directly is faster than computing the full solution; and when the original matrix equation is real, using real arithmetic is faster than using complex arithmetic.

We also did rough comparisons between our scheme *lyapUR* and the very efficient Lyapunov solver *sllyap* in the control library SLICOT [4]. Figure 3.2 was sent to us by Dr. Peter Benner, this comparison was done in Matlab 6.0. From Figure 3.2, we see that *sllyap* is very efficient, the CPU time difference will be larger if the comparison is done in Matlab 5.3.

We did not compare our Matlab code directly with *sllyap* in Matlab because *sllyap* is essentially Fortran code which calls the fine-tuned subroutines in BLAS and LAPACK, while we wrote Fortran 90 code which also called BLAS and LAPACK. The reason for choosing Fortran 90 is that one can program in Fortran 90 quite similarly as in Matlab, this saved a lot of programming time in translating our Matlab codes to Fortran 90 codes.

We compared our Fortran 90 codes and the Fortran 77 *sllyap* in an UltraSPARC II SunOS5.6 workstation with 450 MHz CPU. Figure 3.3 contains one of the many similar results we obtained. From Figure 3.3, we see that our code, even though it is not well tuned, is competitive to the routine from the well tuned, very efficient SLICOT library. We used Fortran 90 features like operator overloading and assumed shape arrays and allocatable arrays for programming simplicity which may cost more CPU time during execution. Also the f77 compiler has higher optimization level than f90 which generally leads to faster executable codes when the f77 programs are fine tuned. Under these circumstances, our code still gives competitive results. This suggests that our modified formulations of the Bartels-Stewart algorithm and the Hammarling method are promising.

Remark 3.1. We observed that Hammarling's method implemented in SLICOT routine *linmeq(3)* is much slower than *sllyap* (which is essentially

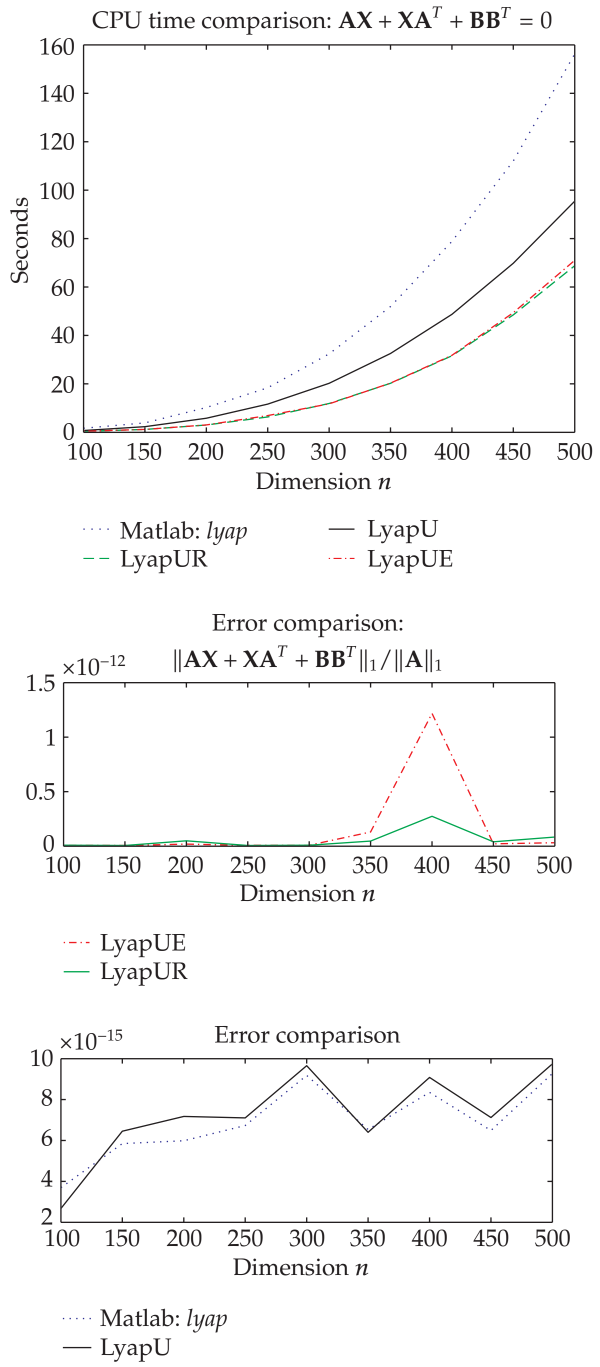


FIGURE 3.1. Performance comparison for the Lyapunov solvers (Matlab 6.0).

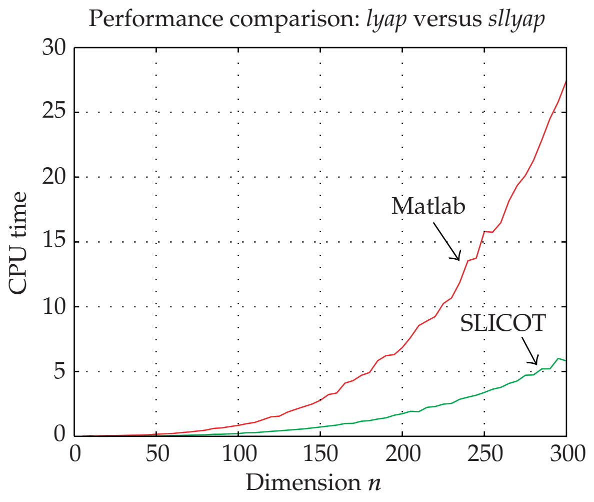
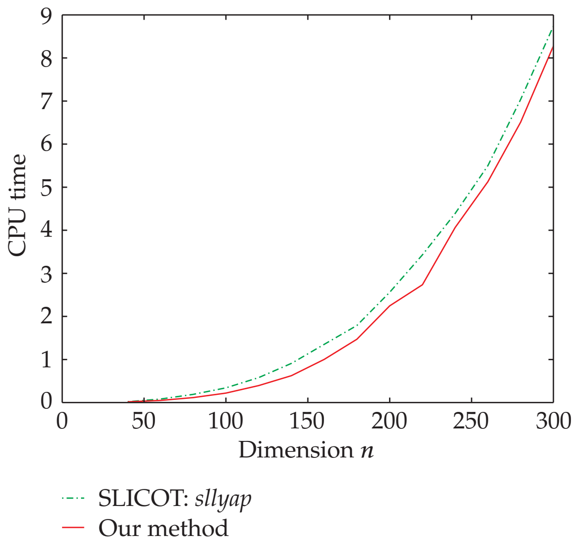


FIGURE 3.2. CPU time comparison: *sllyap* versus Matlab 6.0 *lyap.m*.

linmeq(2)) on Unix workstations. Therefore, we compared our code to the more efficient *sllyap*.

As pointed out in a recent paper by Jonsson and Kågström [14], the Bartels-Stewart algorithm and Hammarling's method carried out explicitly (as done in Matlab, LAPACK, and SLICOT) are mainly level-2 BLAS routines. Recursive algorithms in solving triangular matrix equations (the second stage of the Bartels-Stewart algorithm) are constructed in [14] based on the level-3 BLAS. The formulations can more fully exploit the advantages provided by modern high-performance computer hardware which contain several-level cache memories. Hence, algorithms in [14] are very efficient for triangular matrix equations with large n and should be the choice for large-scale triangular matrix equations.

Our modifications in this section are also mainly of level 2 since our targets are small-to-medium scale matrix equations, this does not become a serious drawback comparing to level-3 routines. For recursive algorithms in [14], it is observed that a faster lowest-level kernel solver (with suitable block size) leads to very efficient solver of triangular matrix equations, what we presented here also contribute to the efficiency of the last-level kernel solver. For models with large dimension n, usually the matrix $\mathbf{A}$ has a banded or a sparse structure, applying the Bartels-Stewart type algorithm becomes impractical because the first stage of the Bartels-Stewart algorithm is Schur decompositions (or Hessenberg-Schur [9]), which cost expensive $O(n^3)$ flops, and the sparse or banded



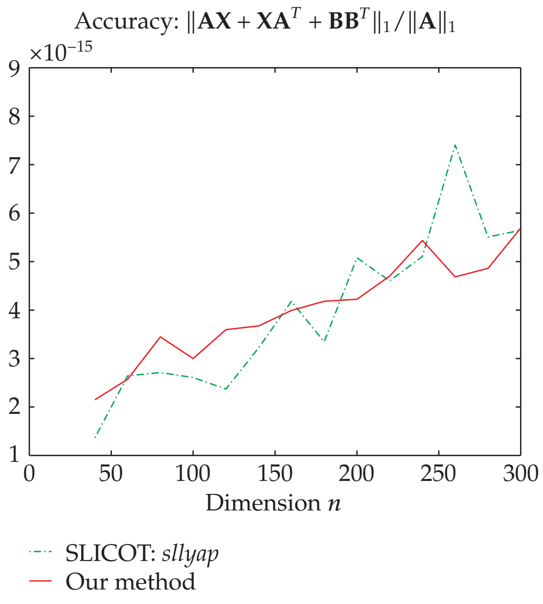


FIGURE 3.3. Performance comparison: *lyapUR* versus *sllyap*.

structure will be destroyed. Hence one usually resorts to iterative projection methods when n is large, and the Bartels-Stewart type algorithms including the ones presented in this section become suitable for the reduced small-to-medium matrix equations.

4. Concluding remarks

In this paper, we revisited the Bartels-Stewart algorithm for Sylvester equations, and the Hammarling algorithm for positive (semi-)definite Lyapunov equations. We proposed column-wise elimination schemes to handle the (2×2)-diagonal blocks, we also constructed a new rank-1 update formula for the Hammarling method. Flop comparison and numerical results show that our modifications improve the performance of the original formulations of these two standard methods. For the comparison with *lyap.m*, our codes are also written in Matlab, hence it is the efficiency of the algorithm reformulation that leads to the superior performance, not because of the programming language. The efficiency of our modified formulation can also be shown when we compared our Fortran 90 code with the Fortran 77 *sllyap*. Our formulations hopefully will enrich the dense methods for small-to-medium scale Sylvester and Lyapunov matrix equations.

Acknowledgments

We wish to thank Dr. Peter Benner for some valuable discussions and for allowing us to use Figure 3.2; Dr. Bo Kågström for sending us his very worth reading recent paper [14]; and Dr. Vasile Sima and Dr. Hongguo Xu for some helpful suggestions about the SLICOT package. We are also grateful to the anonymous referee for the very insightful comments and precise corrections which significantly improved this paper. This work was supported in part by the National Science Foundation (NSF) Grant CCR-9988393.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Pennsylvania, 1999.

[2] A. C. Antoulas and D. C. Sorensen, *Approximation of large-scale dynamical systems: An overview*, Tech. Report TR01-01, Rice University, Texas, 2001.

[3] R. H. Bartels and G. W. Stewart, *Solution of the matrix equation* $AX + XB = C$, Comm. ACM **15** (1972), no. 9, 820–826.

[4] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga, *SLICOT—a subroutine library in systems and control theory*, Applied and Computational Control, Signals, and Circuits, Vol. 1, Birkhäuser Boston, Massachusetts, 1999, pp. 499–539.

[5] P. Benner and E. S. Quintana-Ortí, *Solving stable generalized Lyapunov equations with the matrix sign function*, Numer. Algorithms **20** (1999), no. 1, 75–100.

[6] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, Software, Environments, and Tools, SIAM, Pennsylvania, 1998.

[7] A. R. Ghavimi and A. J. Laub, *Backward error, sensitivity, and refinement of computed solutions of algebraic Riccati equations*, Numer. Linear Algebra Appl. **2** (1995), no. 1, 29–49.

[8] ———, *An implicit deflation method for ill-conditioned Sylvester and Lyapunov equations*, Internat. J. Control **61** (1995), no. 5, 1119–1141.

[9] G. H. Golub, S. Nash, and C. F. Van Loan, *A Hessenberg-Schur method for the problem $AX + XB = C$*, IEEE Trans. Automat. Control **24** (1979), no. 6, 909–913.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Maryland, 1996.

[11] S. J. Hammarling, *Numerical solution of the stable, nonnegative definite Lyapunov equation*, IMA J. Numer. Anal. **2** (1982), no. 3, 303–323.

[12] N. J. Higham, *Perturbation theory and backward error for $AX - XB = C$*, BIT **33** (1993), no. 1, 124–136.

[13] ———, *Accuracy and Stability of Numerical Algorithms*, SIAM, Pennsylvania, 1996.

[14] I. Jonsson and B. Kågström, *Recursive blocked algorithms for solving triangular matrix equations—part I: one-sided and coupled Sylvester-type equations*, ACM Trans. Math. Software **28** (2002), no. 4, 392–415.

[15] A. J. Laub, M. T. Heath, C. C. Paige, and R. C. Ward, *Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms*, IEEE Trans. Automatic Control **32** (1987), no. 2, 115–122.

[16] T. Penzl, *Numerical solution of generalized Lyapunov equations*, Adv. Comput. Math. **8** (1998), no. 1-2, 33–48.

[17] V. Sima, *Algorithms for Linear-Quadratic Optimization*, Monographs and Textbooks in Pure and Applied Mathematics, vol. 200, Marcel Dekker, New York, 1996.

[18] D. C. Sorensen and Y. Zhou, *Direct methods for matrix Sylvester and Lyapunov equations*, Tech. Report TR02, Rice University, Texas, 2002.

[19] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*, Prentice Hall, New Jersey, 1996.

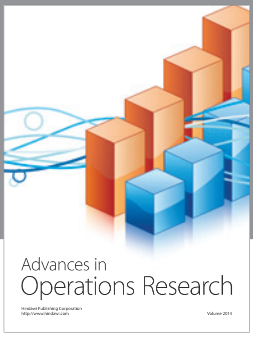
Danny C. Sorensen: Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005-1892, USA

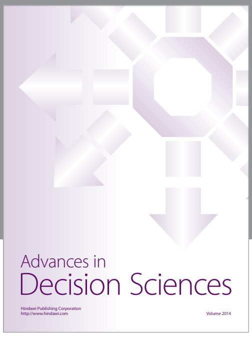
E-mail address: sorensen@caam.rice.edu

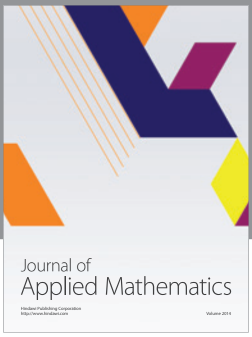
Yunkai Zhou: Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005-1892, USA

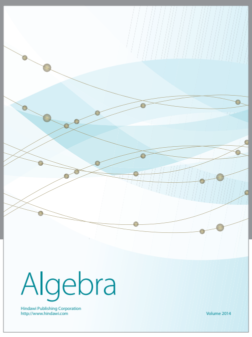
Current address: Argonne National Laboratory, Math and Computer Science Division, Argonne, IL 60439, USA

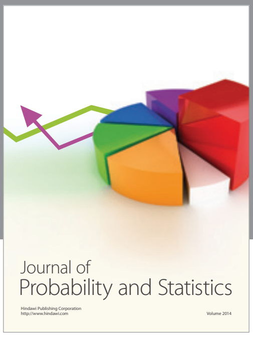
E-mail address: ykzhou@caam.rice.edu


Advances in Operations Research


Advances in Decision Sciences


Journal of Applied Mathematics

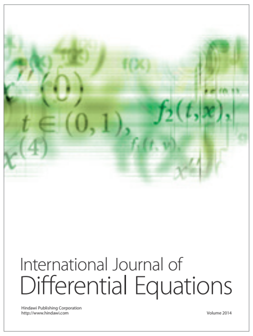

Algebra


Journal of Probability and Statistics

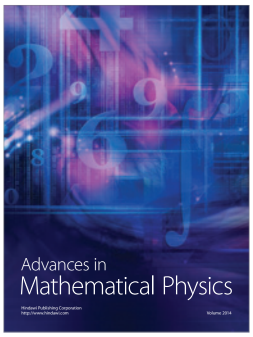
The Scientific World Journal

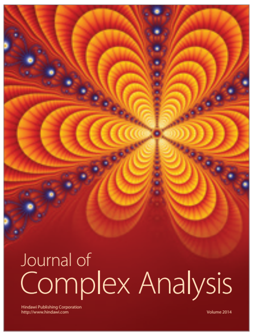
Hindawi

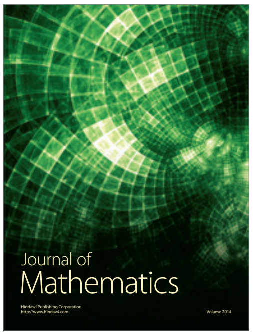
Submit your manuscripts at http://www.hindawi.com

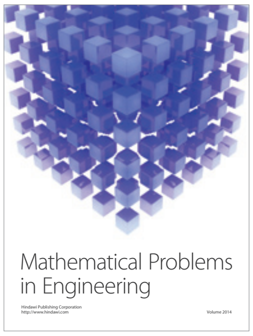

International Journal of Differential Equations

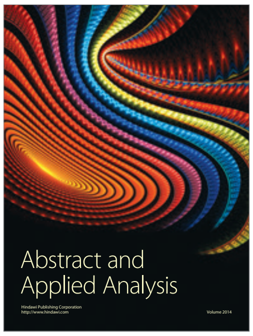
International Journal of Combinatorics

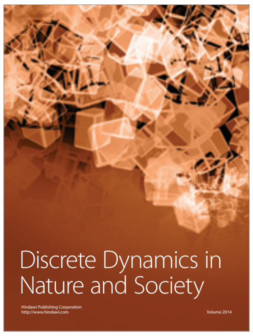

Advances in Mathematical Physics

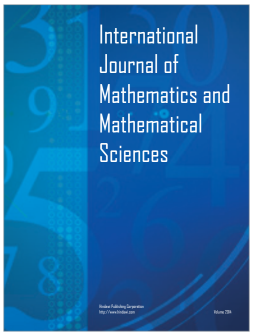

Journal of Complex Analysis


Journal of Mathematics


Mathematical Problems in Engineering

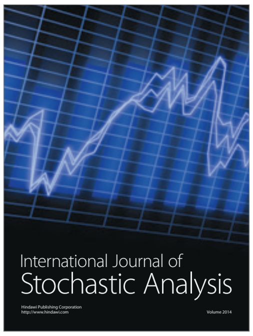

Abstract and Applied Analysis


Discrete Dynamics in Nature and Society


International Journal of Mathematics and Mathematical Sciences

Journal of Discrete Mathematics

Journal of Function Spaces


International Journal of Stochastic Analysis

Journal of Optimization