Peter Benner Enrique S. Quintana-Ortí
Gregorio Quintana-Ortí

Solving Stable Sylvester Equations via Rational Iterative Schemes

Preprint SFB393/04-08

Preprintreihe des Chemnitzer SFB 393

ISSN 1619-7178 (Print) ISSN 1619-7186 (Internet)

SFB393/04-08 **October 2004**

Contents

Author's addresses:

Peter Benner
TU Chemnitz
Fakultät für Mathematik
D-09107 Chemnitz
`benner@mathematik.tu-chemnitz.de`

Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores
Universidad Jaume I
12.071–Castellón, Spain
`{quintana,gquintan}@icc.uji.es`

Abstract

We investigate the numerical solution of stable Sylvester equations via iterative schemes proposed for computing the sign function of a matrix. In particular, we discuss how the rational iterations for the matrix sign function can efficiently be adapted to the special structure implied by the Sylvester equation. For Sylvester equations with factored constant term as those arising in model reduction or image restoration, we derive an algorithm that computes the solution in factored form directly. We also suggest convergence criteria for the resulting iterations and compare the accuracy and performance of the resulting methods with existing Sylvester solvers. The algorithms proposed here are easy to parallelize. We report on the parallelization of those algorithms and demonstrate their high efficiency and scalability using experimental results obtained on a cluster of Intel Pentium Xeon processors.

Keywords. Sylvester equation, matrix sign function, Newton iteration, Newton-Schulz iteration, Halley's method, model reduction, image restoration, parallel algorithms.

1 Introduction

Consider the (continuous-time) *generalized Sylvester equation*

$$AXD + EXB + C \;=\; 0, \tag{1}$$

where $A, E \in \mathbb{R}^{n\times n}$, $B, D \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times m}$, and $X \in \mathbb{R}^{n\times m}$ is the sought-after solution. Equation (1) has a unique solution if and only if $\alpha + \beta \neq 0$ for all $\alpha \in \Lambda(A, E)$ and $\beta \in \Lambda(B, D)$, where $\Lambda(Z, Y)$ denotes the generalized eigenspectrum (or generalized eigenvalues) of a matrix pencil $Z - \lambda Y$. In particular, this property holds for generalized *stable* Sylvester equations, where both $\Lambda(A, E)$ and $\Lambda(B, D)$ lie in the open left half plane. The antistable case, where $\Lambda(A, E)$ and $\Lambda(B, D)$ are contained in the open right half plane, is trivially turned into a stable one by multiplying (1) by -1.

In case $D = I_m$ and $E = I_n$, (1) reduces to the standard *Sylvester equation*

$$AX + XB + C \;=\; 0. \tag{2}$$

Here, the existence and uniqueness of the solution X of (2) is determined by $\Lambda(A)$ and $\Lambda(B)$, the spectra of the corresponding matrices. Also, in case D and E are nonsingular, (1) can be reduced to a standard equation,

$$\tilde{A}X + X\tilde{B} + \tilde{C} = 0, \tag{3}$$

using the transformations $\tilde{A} \leftarrow E^{-1}A$, $\tilde{B} \leftarrow BD^{-1}$, $\tilde{C} \leftarrow E^{-1}CD^{-1}$. The solution X of (1) is not altered by this transformation. However, in practice this transformation may introduce (large) rounding errors in the data and should therefore be avoided.

Sylvester equations have numerous applications in control theory, signal processing, filtering, model reduction, image restoration, decoupling techniques for ordinary and partial differential equations, implementation of implicit numerical methods for ordinary differential equations, and block-diagonalization of matrices; see, e.g., [14, 16, 21, 22, 18, 27, 44] to name only a few references. In some applications, in particular in model reduction using cross-Gramians [2, 23], image restoration [14], and observer design [17], the right-hand side of (2) is given in factored form, $C = FG$, where $F \in \mathbb{R}^{n\times p}$, $G \in \mathbb{R}^{p\times m}$. If $p \ll n, m$ then it can often be observed that the solution also presents a low (numerical) rank [30]. It is therefore beneficial to be able to compute the solution of the Sylvester equation in factorized form directly. Here, we will discuss a new approach based on the sign function method to achieve this goal; see also [8].

Also, $B = A^T$ and $D = E^T$ yields the (continuous-time) *Lyapunov equation* such that everything derived here can be used (and simplified) for this type of equations playing a vital role in many areas of computer-aided control system design (CACSD) [4, 17, 24, 44]. Many of these applications lead to *stable* Sylvester and Lyapunov equations.

Standard solution methods for Sylvester equations of the form (2) are the Bartels-Stewart method [6] and the Hessenberg-Schur method [21, 28], which can be generalized to equations of the form (1), as shown in [22, 26]. These methods are based on transforming the coefficient matrices (A, D, E, and B) to Schur or Hessenberg form and then solving the corresponding linear system of equations directly by a back-substitution process. Therefore, these are classified as *direct methods*. Several iterative schemes to solve Sylvester equations have also been proposed. The focus in this paper is on a special class of iterative schemes to solve stable Sylvester equations that appear in the computation of the matrix sign function. The basic Newton iteration classically used in this context was first proposed for (2) in [42]; see also [7, 36]. It can basically be described by the recursive application of the rational function $f(z) = z + \frac{1}{z}$ to a suitably chosen matrix. Many other polynomial and rational iteration functions can be employed, though. We will investigate the application of some of them to the solution of stable Sylvester equations and will discuss their properties. We will also show how to generalize the classical Newton iteration in order to solve (1). For other methods focusing on large sparse systems see, e.g., [14, 31, 37, 47].

Throughout this paper, we will assume that A, B, C, and D are nonsingular and that $A - \lambda E$, $B - \lambda D$ are stable matrix pencils. This is a very strong assumption as compared to the necessary ones ($A - \lambda C$, $D - \lambda B$ both regular with disjoint spectra) that can be handled by the Bartels-Stewart and Hessenberg-Schur methods. Therefore, the methods developed here can only be applied to a limited set of equations of the forms (1) and (2). On the other hand, so far there is no Bartels-Stewart and Hessenberg-Schur based method to treat the special case of a factored right-hand side. Furthermore, we will see that the iterative schemes here are very well-suited for solving large scale Sylvester equations with generally unstructured coefficient matrices. The proposed algorithms are often more efficient and in general as accurate as the direct methods as will be demonstrated by several MATLAB experiments. Also, the iterative schemes can easily be adapted to parallel computing environments in contrast to the direct methods. The Bartels-Stewart and Hessenberg-Schur methods require, at a first stage, the computation of the Schur form of a matrix/matrix pencil by means of the QR/QZ algorithm. The parallelization of the QR algorithm on distributed memory architectures has so far proved to be a difficult task [34] and the performance results are far from those of more common BLAS-3 operations [29]. The parallelization of the QZ algorithm is even a more ambitious goal and no parallel version is available yet! In contrast, the iterative methods based on the sign function are specially attractive in that they can easily be parallelized and deliver excellent performance on parallel architectures. Similar algorithms have been proposed for the discrete-time version of the Sylvester equation in [11] and for the Lyapunov equation in [10, 9].

The rest of the paper is structured as follows. In Section 2 we describe iterative schemes for the solution of the standard Sylvester equation that are derived from three different iterative schemes for the computation of the matrix sign function, namely, the Newton and Newton-

Schulz iterations, and Halley's method. In section 3 we derive the new factored iteration. A variant of the Newton iteration is then used in Section 4 in order to obtain an algorithm for the generalized Sylvester equation (1). Numerical experiments reporting the numerical accuracy and efficiency as compared to MATLAB and SLICOT Sylvester equation solvers as well as the parallel performance on a cluster of Intel Pentium Xeon processors are given in Section 5. In that section we also give some details on how the Sylvester equation solvers are parallelized using the kernels from ScaLAPACK [12], a parallel linear algebra library for distributed-memory computers. The paper ends by some concluding remarks in Section 6.

2 Iterative Schemes for Solving Stable Sylvester Equations

As the iterative schemes developed in this section are all based on the matrix sign function, we will first review some important properties of this matrix function as well as its relation to Sylvester equations.

2.1 Theoretical background

Consider a matrix $Z \in \mathbb{R}^{n\times n}$ with no eigenvalues on the imaginary axis, and let $Z = S\begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix}S^{-1}$ be its Jordan decomposition. Here, the Jordan blocks in $J^- \in \mathbb{R}^{k\times k}$ and $J^+ \in \mathbb{R}^{(n-k)\times(n-k)}$ contain, respectively, the stable and unstable parts of $\Lambda(Z)$. The *matrix sign function* of Z is defined as $\operatorname{sign}(Z) := S\begin{bmatrix} -I_k & 0 \\ 0 & I_{n-k} \end{bmatrix}S^{-1}$. Many other definitions of the sign function can be given; see [40] for an overview. Some important properties of the matrix sign function are summarized in the next lemma.

Lemma 2.1 Let $Z \in \mathbb{R}^{n\times n}$ with no eigenvalues on the imaginary axis. Then:

a) $\operatorname{sign}(Z)^2 = I_n$, i.e., $\operatorname{sign}(Z)$ is a square root of the identity matrix;

b) $\operatorname{sign}\left(T^{-1}ZT\right) = T^{-1}\operatorname{sign}(Z)\,T$ for all nonsingular $T \in \mathbb{R}^{n\times n}$;

c) if Z is stable, then
$$\operatorname{sign}(Z) = -I_n, \quad \operatorname{sign}(-Z) = I_n.$$

Part a) of the previous lemma suggests that we can apply Newton's root-finding iteration to $Z^2 = I_n$ in order to compute $\operatorname{sign}(Z)$, if the starting point is chosen as Z. Thus, we obtain the Newton iteration for the matrix sign function:

$$Z_0 \leftarrow Z, \quad Z_{k+1} \leftarrow \frac{1}{2}(Z_k + Z_k^{-1}), \quad k = 0, 1, 2, \ldots. \tag{4}$$

Under the given assumptions, the sequence $\{Z_k\}_{k=0}^{\infty}$ converges to $\operatorname{sign}(Z) = \lim_{k\to\infty} Z_k$ [42] with an ultimately quadratic convergence rate. The iteration (4) is driven by the rational function
$$f(z) = \frac{1}{2}(z + \frac{1}{z}),$$
and thus is classified as *rational iterative method* in [39]. As the initial convergence may be slow, the use of acceleration techniques is recommended; e.g., *determinantal scaling* [13] is given by
$$Z_k \leftarrow \frac{1}{c_k}Z_k, \quad c_k = |\det(Z_k)|^{\frac{1}{n}}.$$

Another choice, suggested by Higham [35], is *optimal norm scaling*

$$Z_k \leftarrow \frac{1}{c_k} Z_k, \quad c_k = \sqrt{\frac{\|Z_k\|_2}{\|Z_k^{-1}\|_2}},$$

which has certain minimization properties in the context of computing polar decompositions. In order to avoid the computationally expensive spectral norm one can use instead the Frobenius norm. Another alternative is based on using the inequality $\|Z\|_2 \leq \sqrt{\|Z\|_1 \|Z\|_\infty}$ as an approximation, yielding

$$Z_k \leftarrow \frac{1}{c_k} Z_k, \quad c_k = \left(\frac{\|Z_k\|_1 \|Z_k\|_\infty}{\|Z_k^{-1}\|_1 \|Z_k^{-1}\|_\infty} \right)^{\frac{1}{4}}. \tag{5}$$

Numerical experiments suggest that in the context of solving stable Sylvester equations, where (4) will be applied to stable matrices Z, the *approximate norm scaling* (5) performs much better than determinantal scaling. A theoretical foundation for this observation has yet to be found.

In order to find the relation of the matrix sign function to Sylvester equations, consider the following basic decoupling property of the solutions of Sylvester equations. Namely, if X is a solution of (2), the similarity transformation defined by $\left[\begin{smallmatrix} I_n & X \\ 0 & I_m \end{smallmatrix}\right]$ can be used to block-diagonalize the block upper triangular matrix

$$H = \begin{bmatrix} A & C \\ 0 & -B \end{bmatrix} \tag{6}$$

as follows:

$$\begin{aligned} \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix}^{-1} \begin{bmatrix} A & C \\ 0 & -B \end{bmatrix} \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} &= \\ \begin{bmatrix} I_n & -X \\ 0 & I_m \end{bmatrix} \begin{bmatrix} A & C \\ 0 & -B \end{bmatrix} \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} &= \begin{bmatrix} A & 0 \\ 0 & -B \end{bmatrix}. \end{aligned} \tag{7}$$

Using the matrix sign function of H, the relation given in (7), and Lemma 2.1b), we derive the following expression for the solution of the Sylvester equation (2):

$$\operatorname{sign}(H) = \begin{bmatrix} -I_n & 2X \\ 0 & I_m \end{bmatrix}. \tag{8}$$

This relation forms the basis of the numerical algorithms considered here since it states that we can solve (2) by computing the matrix sign function of H from (6). Therefore, we may apply the iterative schemes proposed for sign function computations in order to solve (2). This was first observed in [42] and re-discovered, e.g., in [7, 36].

In the following, we will review some of the most frequently used iterative schemes for computing the sign function and we will show how to adapt these iterations to the solution of the Sylvester equation. In particular we consider three different rational iteration schemes: the Newton iteration, the Newton-Schulz iteration, and Halley's method. The latter two are obtained from the Padé approximations of the sign function. Specifically, in the corresponding

Padé table, Halley's method is given as the $(1,1)$ entry while Newton-Schulz represents the $(0,1)$ entry [39]. From the computational point of view, it seems that only those two entries (and the $(1,0)$ entry which is just the inverse of the Newton iteration (4)) are interesting. All other entries require higher powers of the iterate which may cause problems due to overflow and rounding errors, while making these iterative schemes very expensive. Nevertheless, some of the other iterations that can be obtained from the Padé approximation of the sign function could also be used to obtain parallel algorithms for computing the matrix sign function [39]. These algorithms exhibit coarse-grain parallelism, but are not considered here for the above-mentioned reasons.

2.2 The Newton iteration

Roberts [42] already observed that the Newton iteration (4), applied to H from (6), can be written in terms of separate iterations for A, B, and C:

$$\begin{array}{lll} A_0 := A, & A_{k+1} := \frac{1}{2}\left(A_k + A_k^{-1}\right), & \\ B_0 := B, & B_{k+1} := \frac{1}{2}\left(B_k + B_k^{-1}\right), & k = 0,1,2,\ldots, \\ C_0 := C, & C_{k+1} := \frac{1}{2}\left(C_k + A_k^{-1}C_kB_k^{-1}\right), & \end{array} \tag{9}$$

thus saving a considerable computational cost and workspace. The iterations for A and B compute the matrix sign functions of A and B, respectively. Hence, because of the stability of A and B and Lemma 2.1 c), we have

$$A_\infty := \lim_{k\to\infty} A_k = \operatorname{sign}(A) = -I_n, \tag{10}$$
$$B_\infty := \lim_{k\to\infty} B_k = \operatorname{sign}(B) = -I_m, \tag{11}$$

and, if we define $C_\infty := \lim_{k\to\infty} C_k$, then $X = C_\infty/2$ is the solution of the Sylvester equation (2).

The iterations for A and B can be implemented in the way suggested by Byers [13], i.e.,

$$Z_{k+1} := Z_k - \frac{1}{2}\left(Z_k - Z_k^{-1}\right), \tag{12}$$

with $Z \in \{A, B\}$. This iteration treats the update of Z_k as a "correction" to the approximation Z_k. This can sometimes improve the accuracy of the computed iterates in the final stages of the iteration, when the limiting accuracy is almost reached. In some ill-conditioned problems, the usual iteration may stagnate without satisfying the stopping criterion while the "defect correction" iteration (12) will satisfy the criterion.

A simple stopping criterion for (9) is obtained from Lemma 2.1c) and (10)–(11), respectively. This suggests to stop the iteration if the relative errors for the computed approximations to the sign functions of A and B are both small enough, i.e., if

$$\max\{\|A_k + I_n\|, \|B_k + I_m\|\} \le \tau, \tag{13}$$

where τ is the tolerance threshold. One might choose $\tau = \gamma\varepsilon$ for the machine precision ε and, for instance, $\gamma = n$ or $\gamma = 10\sqrt{n}$. However, as the terminal accuracy sometimes can not be reached, in order to avoid stagnation it is better to choose $\tau = \sqrt{\varepsilon}$ and to perform 1–3

Algorithm 1 Newton Iteration for the Sylvester Equation.

```
INPUT: Coefficient matrices $(A, B, C) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times n}$ as in (2), tolerance $\tau$ for
    convergence of (9).
OUTPUT: Approximate solution $X$ of (2).
1: while $\max\{\|A + I_n\|_1, \|B + I_m\|_1\} > \tau$ do
2:     Use the LU decomposition or the Gauss-Jordan elimination to compute
           $A_{inv} := A^{-1}, \quad B_{inv} := B^{-1}, \quad W := A_{inv} C B_{inv}.$
3:     Compute the scaling parameter $c$ according to (5) using the information contained in
       $A,\ B,\ C$ (needed for $\|Z_k\|$) and $A_{inv},\ B_{inv},\ W$ (needed for $\|Z_k^{-1}\|$).
4:     Set $A := \frac{1}{2}(\frac{1}{c}A + cA_{inv})$.
5:     Set $B := \frac{1}{2}(\frac{1}{c}B + cB_{inv})$.
6:     Set $C := \frac{1}{2}(\frac{1}{c}C + cW)$.
7: end while
8: Set $X := \frac{1}{2}C$.
```

additional iterations once this criterion is satisfied. Due to the quadratic convergence of the Newton iteration (4), this is usually enough to achieve the attainable accuracy.

The computational cost of (9) is dominated by forming the inverses of A_k and B_k, respectively, and the solution of two linear systems for the sequence C_k. If we consider the factorizations of A_k and B_k to be available as a by-product of the inversion, this adds up to $2(n^3+nm(n+m)+m^3)$ flops (floating-point arithmetic operations) as compared to $2(n+m)^3$ flops for a general Newton iteration step for an $(n+m)\times(n+m)$ matrix. Thus, e.g., if $n = m$, iteration (9) requires only half of the computational cost of the general iteration (4). We would obtain the same computational cost had we computed the inverse matrices by means of a Gauss-Jordan elimination procedure and the next matrix in the sequence C_k as two matrix products. Compared with the traditional matrix inversion procedure, based on the Gaussian elimination (also known as LU factorization), parallel Gauss-Jordan transformations present a better load balance and lead to more efficient parallel algorithms [41]. For $n = m$, the cost for the Bartels-Stewart method can be estimated to be roughly $56n^3$ [27], the cost for the Hessenberg-Schur method is about two thirds of this [44]. With the above considerations we can conclude that 7 steps of (9) match the cost of the Bartels-Stewart method while already 5 steps are more expensive than applying the Hessenberg-Schur method. As usually, 7–10 iterations are needed for convergence, (9) is not so appealing on first sight. On the other hand, (9) is rich in BLAS 3 operations in contrast to the two direct methods. Thus it can be expected that the actual execution times of (9) are at least competitve with the direct methods. This is confirmed by the experiments reported in subsection 5.1.

Algorithm 1 describes an implementation of the sign function method that can be coded in a straightforward manner as a Matlab function.

Algorithm 1 can be implemented using workspace for A, B, and C and a scratch array of size $(n^2 + nm + m^2)$. Altogether, this adds up to workspace of size $2(n^2 + nm + m^2)$ as compared to $2(n + m)^2$ for the general iteration (4). For $n = m$, this saves a workspace for about $2n^2$ real numbers.

2.3 The Newton-Schulz iteration

For a general matrix Z, with no eigenvalues on the imaginary axis, the Newton-Schulz iteration is given by

$$Z_0 := Z, \qquad Z_{k+1} := \frac{1}{2} Z_k \left(3I - Z_k^2\right), \quad k = 0, 1, 2, \ldots. \tag{14}$$

This iteration arises from replacing Z_k^{-1} in (4) by the Schulz iteration for the inverse of a matrix [43] and has been studied for sign function computations, e.g., in [5, 39, 40]. It can be interpreted as the matrix version of recursively applying the "rational" (here: polynomial) function

$$f(z) := \frac{3}{2} z - \frac{1}{2} z^3$$

to the matrix Z. The advantage is that, due to the polynomial function driving the iteration, it is inversion-free and rich in matrix-multiplication operations which delivers high efficiency on modern computers [3, 20, 19, 29]. The drawback is that convergence can only be guaranteed if $\|Z_0^2 - I\| < 1$ for some suitable matrix norm. As $\operatorname{sign}(Z)^2 = I$, for any iteration converging to $\operatorname{sign}(Z)$ there exists an integer k_0 such that $\|Z_{k_0}^2 - I\| < 1$. In that case one can switch to the Newton-Schulz iteration, using Z_{k_0} as starting value. This implies a hybrid algorithm: start with some globally convergent iterative scheme and switch to Newton-Schulz as soon as $\|Z_k^2 - I\| < 1$.

Applying Newton-Schulz to H from (6), the iteration can again be simplified. Let

$$H_k := \begin{bmatrix} A_k & C_k \\ 0 & -B_k \end{bmatrix},$$

then

$$H_k^2 := \begin{bmatrix} A_k^2 & A_k C_k - C_k B_k \\ 0 & B_k^2 \end{bmatrix}. \tag{15}$$

Hence, we obtain again three separate iterations:

$$\begin{aligned} A_0 &:= A, & A_{k+1} &:= \tfrac{1}{2} A_k \left(3I_n - A_k^2\right), \\ B_0 &:= B, & B_{k+1} &:= \tfrac{1}{2} B_k \left(3I_m - B_k^2\right), & & k = 0, 1, 2, \ldots. \\ C_0 &:= C, & C_{k+1} &:= \tfrac{1}{2} \left(-A_k(A_k C_k - C_k B_k) + C_k(3I_m - B_k^2)\right), \end{aligned} \tag{16}$$

The workspace requirements for this iteration are of size $2\max(n,m)^2$ plus the space for A, B, and C. The computational cost is twice as much as that of (9), but for $n = m$ half as much as for the Newton-Schulz iteration for a general $(n+m) \times (n+m)$ matrix. As the Newton iteration, the Newton-Schulz iteration is quadratically convergent such that the higher cost compared to (9) is not compensated by a faster convergence. The implementation of this iteration on parallel computers still has some advantages as matrix products are usually more efficient than matrix inversion via Gaussian elimination or Gauss-Jordan elimination.

Evaluating the criterion $\|Z_k^2 - I\| < 1$ requires some extra computation. If the Newton iteration is used as an initial, globally convergent iteration, then Z_k^2 is not readily available (in contrast to the globally convergent Halley's method described in the next subsection). In that case, one may use the identity $Z_k^2 - I = (Z_k + I)(Z_k - I)$ and the inequality

$$\|Z_k^2 - I\| \leq \|Z_k + I\| \|Z_k - I\|$$

Algorithm 2 Newton-Schulz Iteration for the Sylvester Equation.

```
INPUT: Coefficient matrices $(A,B,C) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times n}$ as in (2), tolerance $\tau$ for
    convergence of (9).
OUTPUT: Approximate solution $X$ of (2).
 1: Use Algorithm 1 to compute $A,\ B,\ C$ with
        $\max\{\|A + I_n\|_1, \|B_k + I_m\|_1\} < \sqrt{2} - 1.$
 2: while $\max\{\|A + I_n\|_1, \|B + I_m\|_1\} > \tau$ do
 3:     Compute $V := 3I_m - B^2$.
 4:     Compute $W := AC - CB$.
 5:     Compute $C := CV$.
 6:     Compute $C := \frac{1}{2}(C - AW)$.
 7:     Compute $W := 3I_n - A^2$,
 8:     Set $A := \frac{1}{2}AW$ and $B := \frac{1}{2}BV$.
 9: end while
10: Set $X := \frac{1}{2}C$.
```

(assuming that $\|\,.\,\|$ is a submultiplicative norm). For stable matrices, $Z_k \to -I$ so that $\|Z_k + I\| \to 0$. Using

$$\|Z_k - I\|_1 = \|Z_k + I - 2I\|_1 \le \|Z_k + I\|_1 + 2,$$

a sufficient criterion to ensure that $\|Z_k^2 - I\|_1 < 1$ is given by $\|Z_k + I\|_1 < \sqrt{2} - 1$. It has the advantage that no extra norm computation is needed, but it might be too conservative occasionally. This can be used to measure the convergence of the iteration for the sequences of A_k and B_k without squaring these matrices. As the iteration for the sequences of A_k and B_k are the Newton-Schulz iterations for A and B, respectively, if $\max\{\|A_k + I\|_1, \|B_k + I\|_1\} < \sqrt{2} - 1$ convergence of these iterations is guaranteed. This also implies convergence of the sequence for C_k. This saves two matrix multiplications per Newton iteration, but may lead to more Newton steps than necessary before switching to the Newton-Schulz iteration as this estimate often is conservative. In a variety of numerical tests, though, only one additional Newton iteration was necessary using this relaxed criterion.

The Newton-Schulz method for Sylvester equations can be implemented as described in Algorithm 2. The algorithmic description is oriented towards an efficient use of the BLAS-3 routine for matrix multiplication and re-use of workspace.

2.4 Halley's method

As an example of a higher order rational iterative scheme we consider *Halley's method* [33, 25]. The same ideas employed here can be used to investigate other high-order schemes as the basic property that yields the simplification of the iteration scheme is the block-triangularity of H in (6) and the fact that this property is preserved by matrix multiplication and inversion.

For a general matrix Z, with no eigenvalues on the imaginary axis, Halley's iteration is given by

$$Z_0 \ := \ Z, \qquad Z_{k+1} := Z_k \left(3I + Z_k^2\right)\left(I + 3Z_k^2\right)^{-1}, \quad k = 0, 1, 2, \ldots, \tag{17}$$

which is based on the rational iteration function

$$f(z) := \frac{z^3 + 3z}{3z^2 + 1}.$$

As an instance of the globally convergent Padé approximations (i.e., the (m, k) entries of the Padé table for which $m \geq 1$ and $k \in \{m, m-1\}$; see [39, Theorem 3.3]), this iteration can also be used as a method to generate a starting value for the Newton-Schulz iteration. On the other hand, Halley's method is cubically convergent so that only very few iterations are needed once $\|Z_k^2 - I\| < 1$.

Applied to the Sylvester equation (2) it is again possible to write (17) in terms of Halley's iterations for the sequences A_k and B_k, and a separate iteration for the sequence C_k. As in the Newton-Schulz case, we need the square of the current iterate H_k obtained by applying (17) to H from (6). This was already shown in (15). Using

$$(I + 3H_k^2)^{-1} \;=\; \begin{bmatrix} (I_n + 3A_k^2)^{-1} & -3(I_n + 3A_k^2)^{-1}(A_kC_k - C_kB_k)(I_m + 3B_k^2)^{-1} \\ 0 & (I_m + 3B_k^2)^{-1} \end{bmatrix},$$

and performing some elementary manipulations we obtain the Halley iteration for (2) as

$$\begin{array}{lll} A_0 := A, & A_{k+1} := A_k\left(3I_n + A_k^2\right)\left(I_n + 3A_k^2\right)^{-1}, & \\ B_0 := B, & B_{k+1} := B_k\left(3I_m + B_k^2\right)\left(I_m + 3B_k^2\right)^{-1}, & k = 0, 1, 2, \ldots. \\ C_0 := C, & C_{k+1} := ((A_k - 3A_{k+1})(A_kC_k - C_kB_k) + & \\ & \qquad C_k\left(3I_m + B_k^2\right))\left(I_m + 3B_k^2\right)^{-1}, & \end{array} \tag{18}$$

Given scratch arrays U, V, W, where U, V are of size $(\max(n, m))^2$ and W is of size nm, this iteration can be implemented as in Algorithm 3 where, for convenience, we have provided the values of the computed quantities in terms of (18) as comments.

The procedure given in Algorithm 3 requires $\frac{20}{3}n^3 + 6n^2m + 4nm^2 + \frac{20}{3}m^3$ flops if $n < m$. Otherwise the computations can be rearranged so that the cost becomes $\frac{20}{3}m^3 + 4n^2m + 6nm^2 + \frac{20}{3}n^3$. This implementation requires a scratch space of size $n^2 + 2(\max(m, n))^2 + \max(n, m)m + nm$ plus the space for A, B, and C. However, a tradeoff between computational cost and scratch space is possible so that a larger workspace could be used to reduce the number of actual flops.

The computational savings compared to applying (17) to H from (6) are not as significant as in the Newton-Schulz case. The full scheme would require $23\frac{1}{3}n^3$ flops if $n = m$ and, in general, a workspace of size $3(n + m)^2$. If $n = m$, Algorithm 1 requires a workspace of size $6n^2$ as compared to $12n^2$ for the full scheme so that half this space can be saved.

3 Factorized Solution of Stable Sylvester Equations

Here we consider the Sylvester equation (2) with factorized right-hand side, that is,

$$AX + XB + FG, \qquad F \in \mathbb{R}^{n \times p}, \quad G \in \mathbb{R}^{p \times m}. \tag{19}$$

Algorithm 3 Halley's Method for the Sylvester Equation.

```
INPUT: Coefficient matrices $(A, B, C) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times n}$ as in (2), tolerance $\tau$ for
    convergence of (9).
OUTPUT: Approximate solution $X$ of (2).
 1: while $\max\{\|A + I_n\|_1, \|B + I_m\|_1\} > \tau$ do
 2:   $U := A^2$                    % $= A_k^2$.
 3:   $V := I_n + 3U$               % $= I_n + 3A_k^2$.
 4:   $U := 3I_n + U$               % $= 3I_n + A_k^2$.
 5:   $U := UV^{-1}$                % $= (3I_n + A_k^2)(I_n + 3A_k^2)^{-1}$.
 6:   $V := A$                      % $= A_k$.
 7:   $A := VU$                     % $= A_{k+1}$.
 8:   $U := VC$                     % $= A_k C_k$.
 9:   $U := U + CB$                 % $= A_k C_k - C_k B_k$.
10:   $V := V - 3A$                 % $= A_k - 3A_{k+1}$.
11:   $W := VU$                     % $= (A_k - 3A_{k+1})(A_k C_k + C_k B_k)$.
12:   $U := B^2$                    % $= B_k^2$.
13:   $V := I_m + 3U$               % $= I_m + 3B_k^2$.
14:   $V := V^{-1}$                 % $= (I_m + 3B_k^2)^{-1}$.
15:   $U := 3I_m + U$               % $= 3I_m + B_k^2$.
16:   $U := UV$                     % $= (3I_m + B_k^2)(I_m + 3B_k^2)^{-1}$.
17:   $W := WV$                     % $= (A_k - 3A_{k+1})(A_k C_k + C_k B_k)(I_m + 3B_k^2)^{-1}$.
18:   $B := BU$                     % $= B_{k+1}$.
19:   $V := CU$                     % $= C_k(3I_m + B_k^2)(I_m + 3B_k^2)^{-1}$.
20:   $C := V + W$                  % $= C_{k+1}$.
21: end while
22: Set $X := \frac{1}{2}C$.
```

Applying (9) to (19) the iterations for A_k and B_k remain unaltered, while the iteration for C_k can be rewritten as follows:

$$\begin{aligned} F_0 &:= F, \quad F_{k+1} := \left[\, F_k,\ A_k^{-1}F_k \,\right], \\ G_0 &:= G, \quad G_{k+1} := \begin{bmatrix} G_k \\ G_k B_k^{-1} \end{bmatrix}. \end{aligned}$$

Although this iteration is much cheaper during the initial steps if $p \ll n, m$, this advantage is lost in latter iterations since the number of columns in F_{k+1} and the number of rows in G_{k+1} is doubled in each iteration step. This can be avoided by applying a similar technique as used in [10] for the factorized solution of Lyapunov equations. Let $F_k \in \mathbb{R}^{n\times p_k}$ and $G_k \in \mathbb{R}^{p_k\times m}$. We first compute a rank-revealing QR (RRQR) factorization [15] of G_{k+1} as defined above; that is,

$$\begin{bmatrix} G_k \\ G_k B_k^{-1} \end{bmatrix} = UR\Pi_G, \quad R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where U is orthogonal, Π_G is a permutation matrix, and R is upper triangular with $R_1 \in \mathbb{R}^{r\times m}$ of full row-rank. Then, we compute a RRQR factorization of $F_{k+1}U$:

$$\left[\, F_k,\ A_k^{-1}F_k \,\right] U = VT\Pi_F, \quad T = \begin{bmatrix} T_1 \\ 0 \end{bmatrix},$$

Algorithm 4 Factorized Newton Iteration for the Sylvester Equation.

```
INPUT: Coefficient matrices $(A, B, F, G) \in \mathbb{R}^{n\times n}\times\mathbb{R}^{m\times m}\times\mathbb{R}^{n\times p}\times\mathbb{R}^{p\times m}$ as in (19), tolerances
    $\tau_1$ for convergence of (9) and $\tau_2$ for rank detection.
OUTPUT: Numerical full-rank factors $Y, Z$ of the solution $X$ of (19).
1: while $\max\{\|A + I_n\|_1, \|B + I_m\|_1\} > \tau_1$ do
2:     Use the LU decomposition or the Gauss-Jordan elimination to compute
           $A_{inv} := A^{-1}, \quad B_{inv} := B^{-1}, \qquad c := (\det(A)\det(B))^{-\frac{1}{n+m}}.$
3:     Compute the RRQR $\begin{bmatrix} G/\sqrt{c} \\ \sqrt{c}GB^{-1} \end{bmatrix} =: U \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \Pi_G$, with $U$ orthogonal, $\Pi_G$ a permutation
       matrix, $R_1 \in \mathbb{R}^{r\times m}$ upper triangular of full row-rank.
4:     Compute the RRQR $\begin{bmatrix} F/\sqrt{c}, & \sqrt{c}A^{-1}F \end{bmatrix} U = V \begin{bmatrix} T_1 \\ 0 \end{bmatrix} \Pi_F$, with $V$ orthogonal, $\Pi_F$ a
       permutation matrix, $T_1 \in \mathbb{R}^{t\times 2p}$ upper triangular of full row-rank.
5:     Partition $V = [\, V_1, \; V_2 \,]$, $V_1 \in \mathbb{R}^{n\times t}$, and compute $[\, T_{11}, \; T_{12} \,] := T_1\Pi_F$, where $T_{11} \in \mathbb{R}^{t\times r}$.
6:     Set
           $F := V_1T_{11}, \quad G := R_1\Pi_G, \quad p := r.$
7:     Set
           $A := \frac{1}{2}(\frac{1}{c}A + cA_{inv}), \quad B := \frac{1}{2}(\frac{1}{c}B + cB_{inv}).$
8: end while
9: Set $Y := \frac{1}{\sqrt{2}}F$, $Z := \frac{1}{\sqrt{2}}G$.
```

where V is orthogonal, Π_F is a permutation matrix, and T is upper triangular with $T_1 \in \mathbb{R}^{t\times 2p_k}$ of full row-rank. Partitioning $V = [\, V_1, \; V_2 \,]$, with $V_1 \in \mathbb{R}^{n\times t}$, and computing

$$[\, T_{11}, \; T_{12} \,] := T_1\Pi_F, \qquad T_{11} \in \mathbb{R}^{t\times r},$$

we then obtain as the new iterates

$$F_{k+1} := V_1T_{11}, \qquad G_{k+1} := R_1\Pi_G.$$

Then from C_{k+1} in (9), we have

$$C_{k+1} = F_{k+1}G_{k+1}.$$

Setting

$$Y := \frac{1}{\sqrt{2}} \lim_{k\to\infty} F_k, \qquad Z := \frac{1}{\sqrt{2}} \lim_{k\to\infty} G_k,$$

we obtain the solution X of (2) in factored from $X = YZ$. If X has low numerical rank, the factors Y and Z will have the a low number of columns and rows, respectively, and the storage and computation time needed for the factorized solution will be much lower than that of the original iteration (9). The resulting algorithm is summarized in Algorithm 4, where for ease of notation, we used determinantal scaling—implementing approximate norm scaling is a bit involved here as the C_k-iterate is not available explicitly. A relaxed approximate norm scaling based on ignoring the contributions of F_kC_k in (5) can be used, see Example 2 in subsection 5.1.

Given that $r \ll m, n$ for all the iterates (that is, the solution X has low numerical rank), the cost of Algorithm 4 is $2(n^3 + m^3) + \mathcal{O}((n + m)^2)$ flops, where the cubic part comes from computing the matrix inverses either via the Gaussian elimination or Gauss-Jordan elimination. The required workspace also depends on the numerical rank of X and is about $2(n^2 + m^2 + 2(n+m)r)$ real numbers.

Certainly, variants of Algorithms 4 based on the Newton-Schulz or Halley's method can be derived but, as no gain in efficiency can be expected regarding the numerical results given in Section 5, we omit the tedious details of such an algorithm here.

4 Iterative Schemes for Solving Generalized Stable Sylvester Equations

As outlined in the Introduction, Equation (1) can be solved by first transforming it into a standard Sylvester equation, and then applying any of the the methods presented in Section 2. This is equivalent to computing the matrix sign function of

$$\tilde{H} = \begin{bmatrix} \tilde{A} & \tilde{C} \\ 0 & -\tilde{B} \end{bmatrix} = \begin{bmatrix} E^{-1}A & E^{-1}CD^{-1} \\ 0 & -BD^{-1} \end{bmatrix}.$$

However, we can also use the equivalence of $\tilde{H} - \lambda I_{n+m}$ to

$$H - \lambda K := \begin{bmatrix} A & C \\ 0 & -B \end{bmatrix} - \lambda \begin{bmatrix} E & 0 \\ 0 & D \end{bmatrix}, \tag{20}$$

given by

$$H - \lambda K = K(\tilde{H} - \lambda I_{n+m})L, \quad \text{where} \quad K = \begin{bmatrix} E & 0 \\ 0 & I_m \end{bmatrix}, \quad L = \begin{bmatrix} I_n & 0 \\ 0 & D \end{bmatrix}.$$

Now, from (8) we know that $\operatorname{sign}\left(\tilde{H}\right) = \left[\begin{smallmatrix} -I_n & 2X \\ 0 & I_m \end{smallmatrix}\right]$ so that we can compute the solution of the generalized Sylvester equation by applying (9) to $\tilde{H}$. In doing so, we obtain in the first step (omitting scaling)

$$\tilde{H}_1 = \frac{1}{2}(\tilde{H} + \tilde{H}^{-1}) = \frac{1}{2}(K^{-1}HL^{-1} + LH^{-1}K) = K^{-1}\left(\frac{1}{2}(H + KLH^{-1}KL)\right)L^{-1}.$$

Repeating this calculation and denoting $H_0 := H$, $K_0 := K$, we obtain

$$H_{k+1} := \frac{1}{2}(H_k + KLH_k^{-1}KL), \quad k = 1, 2, \ldots, \tag{21}$$

so that $H_k = K\tilde{H}_kL$. Finally, taking limits on both sides, yields

$$H_\infty := \lim_{k\to\infty} H_k = K \operatorname{sign}\left(\tilde{H}\right) L = \begin{bmatrix} -E & 2EXD \\ 0 & D \end{bmatrix}, \tag{22}$$

so that $X = \frac{1}{2}E^{-1}H_{12}D^{-1}$, where H_{12} denotes the upper right $n \times m$-block of H_∞.

The iteration (21) is an instance of the *generalized Newton iteration* formulated in [25] for a general matrix pair $Z - \lambda Y$, with Y nonsingular:

$$Z_0 := Z, \qquad Z_{k+1} := \left(Z_k + YZ_k^{-1}Y\right), \quad k = 0, 1, 2, \ldots. \tag{23}$$

Using the block-triangular structure of the involved matrix pencil, we obtain from (21) the *generalized Newton iteration* for the solution of the generalized Sylvester equation (1):

$$\begin{array}{lll} A_0 := A, & A_{k+1} := \frac{1}{2}\left(A_k + EA_k^{-1}E\right), & \\ B_0 := B, & B_{k+1} := \frac{1}{2}\left(B_k + DB_k^{-1}D\right), & k = 0,1,2,\ldots . \\ C_0 := C, & C_{k+1} := \frac{1}{2}\left(C_k + EA_k^{-1}C_kB_k^{-1}D\right), & \end{array} \tag{24}$$

At convergence, the solution of (1) is then obtained by solving the linear matrix equation

$$EXD = \frac{1}{2}\lim_{k\to\infty} C_k.$$

From (22) we have

$$\lim_{k\to\infty} A_k = -E, \quad \lim_{k\to\infty} B_k = -D,$$

which suggests the stopping criterion

$$\max\left\{\frac{\|A_k + E\|_1}{\|E\|_1}, \frac{\|B_k + D\|_1}{\|D\|_1}\right\} \leq \tau, \tag{25}$$

where again, τ is the tolerance threshold.

Several scaling strategies for accelerating this iteration can be used. Similarly to Section 2, rather than using the determinantal scaling as suggested for (23) in [25], we use again a norm scaling, based on equalizing the spectral norm of the two addends in (23) and approximating the spectral norm by the geometric mean of the 1-norm and the maximum norm. For (21) we thus obtain

$$c_k := \left(\frac{\|H_k\|_1\|H_k\|_\infty}{\|KLH_k^{-1}KL\|_1\|KLH_k^{-1}KL\|_\infty}\right)^{\frac{1}{4}}. \tag{26}$$

This quantity can be readily computed from the intermediate results in (24).

One step of iteration (24) involves solving two general linear systems, and computing four matrix products. This adds up to $\frac{14}{3}n^3+2n^2m+2nm^2+\frac{14}{3}m^3$ flops. For $n = m$, we obtain $\frac{40}{3}n^3$ flops. The estimates given in [26] for the Bartels-Stewart method are $144n^3$ flops and $106n^3$ flops for the Hessenberg-Schur method. Hence, about 8 iterations of (24) are as expensive as solving (1) by the Hessenberg-Schur method while about 11 iterations are equivalent in cost to that of the Bartels-Stewart method. The usual observation for iteration (24) combined with the scaling as suggested in (26) is that 7–10 iterations are required for convergence of fairly well-conditioned examples. As the computational kernels of (24) can usually be implemented in a much more efficient way than the QZ algorithm, which is the basis of the codes described in [26], it is expected that (24) achieves a much higher performance than the Bartels-Stewart and Hessenberg-Schur methods, especially when implemented on parallel computers.

Algorithm 5 is implements a BLAS-3 variant with efficient memory re-usage for (24). The scratch space necessary for the iteration includes T of size n^2, U of size m^2, and V, W, each of size $(\max(n,m))^2$.

A factorized iteration for generalized Sylvester equations with right-hand side $C = FG$ can be derived in complete analogy to Section 3 defining

$$\begin{array}{rcl} F_0 & := & F, \quad F_{k+1} := \left[\,F_k,\ EA_k^{-1}F_k\,\right], \\ G_0 & := & G, \quad G_{k+1} := \left[\begin{array}{c} G_k \\ G_kB_k^{-1}D \end{array}\right]. \end{array}$$

Algorithm 5 Generalized Newton Iteration for the Generalized Sylvester Equation.

```
INPUT: Coefficient matrices $A, E \in \mathbb{R}^{n\times n}$, $B, D \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{m\times n}$ as in (1), tolerance $\tau$
    for convergence of (24).
OUTPUT: Approximate solution $X$ of (1).
 1: while $\max\{\frac{\|A+E\|_1}{\|E\|_1}, \frac{\|B+D\|_1}{\|D\|_1}\} > \tau$ do
 2:     Use the LU decomposition or the Gauss-Jordan elimination to compute
            $T := EA^{-1}, \quad U := B^{-1}D.$
 3:     Compute $V := CU$.
 4:     Compute $W := TV$.
 5:     Compute $V := TE$, set $T := V$.
 6:     Compute $V := DU$, set $U := V$.
        {Now $T$, $U$, $W$ contain the blocks of the second addend in (21).}
 7:     Compute $c$ as in (26) from $A$, $B$, $C$, $T$, $U$, $W$.
 8:     Compute $A := \frac{1}{2}(\frac{1}{c}A + cT)$.
 9:     Compute $B := \frac{1}{2}(\frac{1}{c}B + cU)$.
10:     Compute $C := \frac{1}{2}(\frac{1}{c}C + cW)$.
11: end while
12: Solve $EXD = \frac{1}{2}C$.
```

We leave the details of such an iteration to the interested reader.

5 Experimental Results

In this section we analyze the accuracy and parallel performance of the various Sylvester equation solvers proposed in this paper.

5.1 Accuracy and efficiency of the Sylvester equation solvers

This subsection briefly analyzes the accuracy of the Sylvester equation solvers based on the sign function. For this purpose, we implemented MATLAB functions according to the Algorithms 1–5. We compare these functions with the Bartels-Stewart method as implemented in the MATLAB function `lyap` from the Control Toolbox. For completeness, we also include in our comparison the Hessenberg-Schur method. An implementation of this method is contained in SLICOT and can be called from MATLAB via a mex gateway function as routine `slsylv`; see [45] for details.

In practice, the accuracy has to be related with the conditioning of the problem. In particular, the conditioning of the Sylvester equation is given by the sensitivity to perturbations in the data, and can be measured using the distance between the spectra of the matrix pencils $A - \lambda E$ and $B - \lambda D$. This distance can be estimated using an approximation of the function $\mathrm{Dif}\,[(A, E), (B, D)]$ [38]. In the standard case of the equation this simplifies to the more well-known separation function, $\mathrm{Sep}\,[A, B]$.

In the evaluation we borrow examples from [1, 45]. All the experiments presented here were performed on an Intel Pentium M processor at 1.4 GHz with 512 MBytes of RAM using

MATLAB Version 6.5.1 (R13) and the MATLAB Control Toolbox Version 5.2.1. MATLAB uses IEEE double-precision floating-point arithmetic with machine precision $\varepsilon \approx 2.2204 \times 10^{-16}$.

Example 1 [45, Example 7]. For the standard Sylvester equation we employ square matrices A, B, and C generated as follows. First, we construct

$$\begin{array}{rcll} \hat{A} & = & \operatorname{diag}\left(-1, -a, -a^2, \ldots, -a^{n-1}\right), & a > 1, \\ \hat{B} & = & \operatorname{diag}\left(-1, -b, -b^2, \ldots, -b^{n-1}\right), & b > 1, \\ \hat{C} & = & \operatorname{diag}\left(1, 2, 3, \ldots, n\right), & \end{array}$$

where the parameters a and b regulate the distribution of the spectra of A and B, respectively, and therefore their separation. The entries of the solution matrix to $\hat{A}\hat{X} + \hat{X}\hat{B} + \hat{C} = 0$ are then given by

$$\hat{X}_{ij} = \frac{\hat{C}_{ij}}{\hat{A}_{ii} + \hat{B}_{jj}}.$$

In a second step, we employ a transformation matrix $T \in \mathbb{R}^{n \times n}$ defined as

$$T = H_2 S H_1,$$

where

$$\begin{array}{rcllr} H_1 & = & I_n - \frac{2}{n} h_1 h_1^T, & & h_1 = [1, 1, \ldots, 1]^T, \\ H_2 & = & I_n - \frac{2}{n} h_2 h_2^T, & & h_2 = \left[1, -1, \ldots, (-1)^{n-1}\right]^T, \\ S & = & \operatorname{diag}\left(1, s, \ldots, s^{n-1}\right), & & s > 1, \end{array}$$

to transform the matrices of the equation as

$$A = T^{-T} \hat{A} T^T, \quad B = T \hat{B} T^{-1}, \quad C = T^{-T} \hat{C} T^{-1}, \quad \text{and} \quad X = T^{-T} \hat{X} T^{-1}.$$

The scalar s is used here to regulate the conditioning of T. In this example we set the parameters to a=1.03, b=1.008, and s=1.001. With increasing dimension n, the condition number of the Sylvester equation increases merely due to an increase in the norms of A and B rather than a decrease of the separation number. This explains the loss of accuracy of all considered methods for growing n. Figure 1 shows the accuracy and execution times of MATLAB `lyap` function, SLICOT `slsylv` function, as well as the Newton, Newton-Schulz, and Halley iterations. In this example, the accuracy of the Bartels-Stewart and Hessenberg-Schur methods is slightly better than that of the sign function-based solvers. While for the Newton iteration, the accuracy deteriorates only slightly, Halley's method is much more affected by roundoff and produces almost useless results for large n. The execution times for the Newton and Newton-Schulz iterations are astonishing as their pure MATLAB implementations even outperform `slsylv` which is based on a compiled and optimized Fortran 77 code. This demonstrates a high potential of the sign function method, in particular when implemented via the Newton and Newton-Schulz iterations, to provide a highly efficient Sylvester solver. The high execution time of Halley's method is partially explained by the large number of iterations (see Figure 2) needed for convergence as the higher complexity per step is not compensated by the cubic convergence rate—as a matter of fact, cubic convergence cannot be claimed for this example. The observed convergence rate is linear with constant $\frac{1}{3}$, and the iteration stagnates without reaching the same level of accuracy as the Newton and Newton-Schulz iterations.

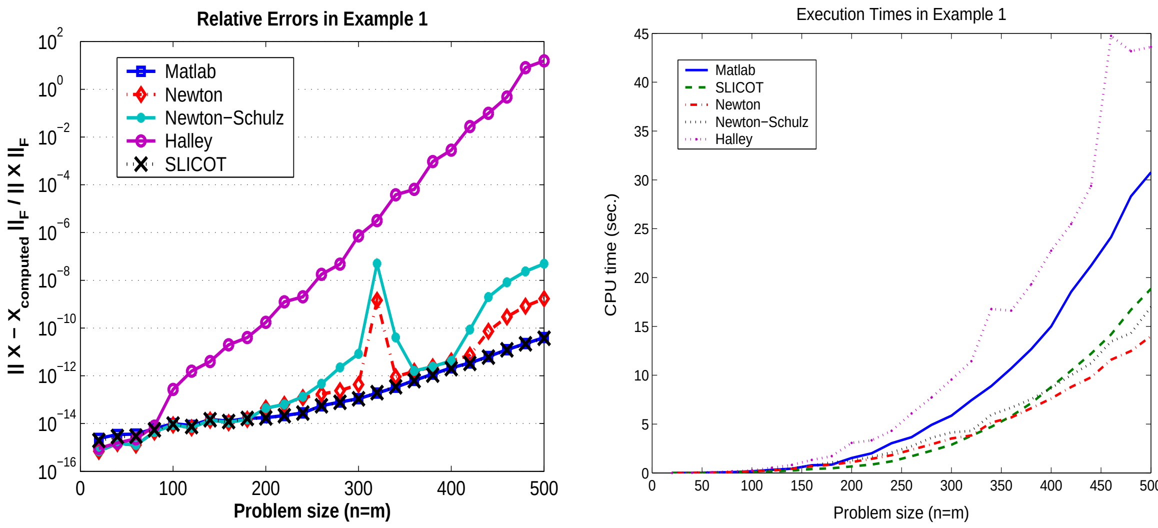


Figure 1: Relative errors and execution times of the Sylvester equation solvers applied to Example 1.

The left plot in Figure 2 shows the iteration numbers for the three different iterations considered for solving the Sylvester equation for $n = 500$ together with the achieved values of the stopping criterion. Additionally, we show also the values for the Newton iteration (9) with determinantal scaling (denoted `Newton (det)` in the figure). In this example, determinantal scaling requires one iteration more in order to reach the same value of the stopping criterion with scaling as in (5). Within the Newton-Schulz iteration, switching to the Schulz iteration is only done in the last two iterations.

Example 2. A possible application of the factored Newton iteration for Sylvester equations is the computation of the *cross-Gramian* of a linear time-invariant system

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t), \tag{27}$$

which is defined as the solution of the Sylvester equation

$$AX + XA + BC = 0. \tag{28}$$

The cross-Gramian contains information about certain properties of the linear system [23] and can also be used for model reduction [2].

We apply Algorithm 4 to (28) for a system described in [1, Example 4.2] which comes from a model for heat control in a thin rod. The physical process is modeled by a linear-quadratic optimal control problem for the instationary 1D heat equation. Semi-discretization in space using finite elements leads to a system of the form (27) with $A = -M^{-1}K$ where M and K are the mass matrix and stiffness matrix, respectively, of the finite element approximation. Mesh refinement leads to systems of different orders n. The other parameters in this example are set to $a = 0.01$, $b = 2$, $c = 1$, $\beta_1 = 0$, $\beta_2 = 0.1$, $\gamma_1 = 0.9$, $\gamma_2 = 1$. We compare the Bartels-Stewart and Hessenberg-Schur methods with Algorithm 4. There are no variants of the direct methods that compute the factored solution directly.

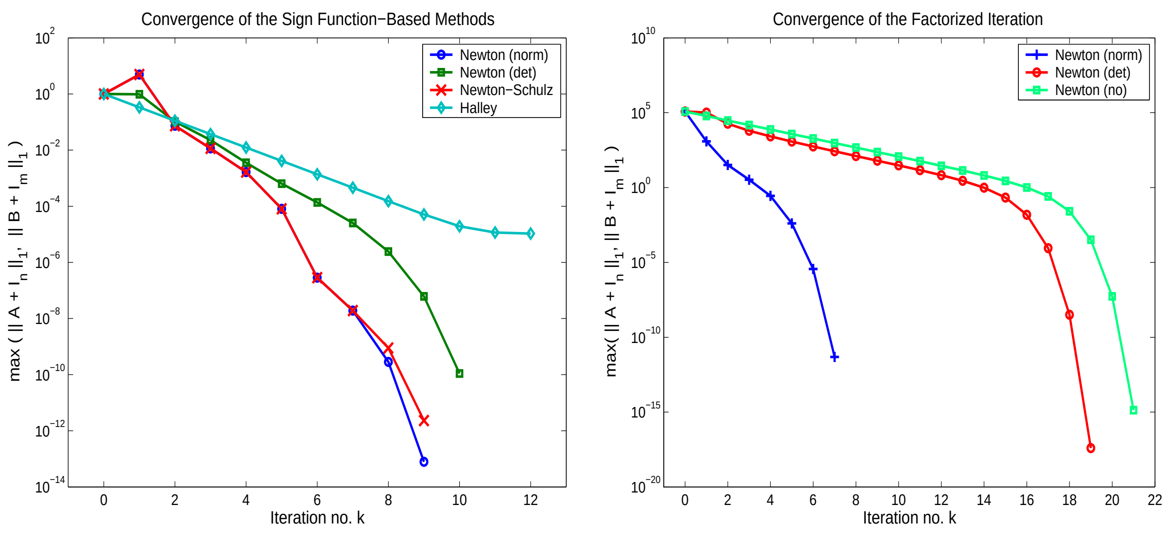


Figure 2: Iteration numbers of the iterative Sylvester equation solvers in Example 1 ($n = 500$, left) and Example 2 (right).

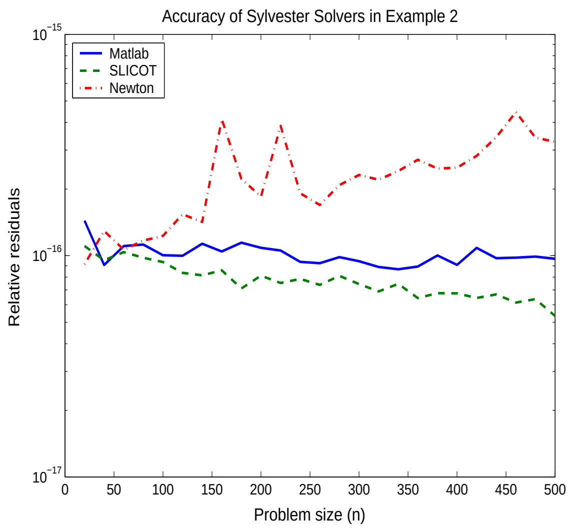
Figure 3 shows the accuracy and execution times of the solvers for various dimensions. As an exact solution is not known in this case, we show the relative residuals

$$\frac{\|A\hat{X} + \hat{X}A + BC\|_F}{2\|A\|_F\|\hat{X}\|_F + \|B\|_F\|C\|_F},$$

where $\hat{X}$ denotes the computed solution. All three methods give a relative residual as small as expected from a numerically backward stable method. It should be noted, though, that for the comparison we have to build $\hat{X} = \hat{Y}\hat{Z}$ with the computed factors returned by Algorithm 4, thereby giving away the advantage resulting from working with the factors directly.

The execution times show a fair advantage of the new algorithm even when only implemented as a straightforward Matlab function. For the largest example in the test set ($n = 500$), Algorithm 4 is three times faster than `slsylv` and five times faster than `lyap`. The gains would be even greater when taking into account that the B_k-iteration in Algorithm 4 is not necessary for cross-Gramian computation (see [8]).

An interesting aspect of this example is that for increasing n, the separation becomes smaller. As this affects the convergence of the Newton iteration for the sign function, we test different scaling strategies: the determinantal scaling (`Newton (det)`), no scaling (`Newton (no)`), and a modification of approximate scaling (5) (`Newton (norm)`) that does not require the unavailable C_k-iterate. The modification is merely a simplification, which basically ignores the contributions of C_k and $A_k^{-1}C_kB_k^{-1}$ needed for computing the scaling factor as in (5). As can be observed in the right plot in Figure 2, this modified norm scaling outperforms determinantal scaling and no scaling by far.



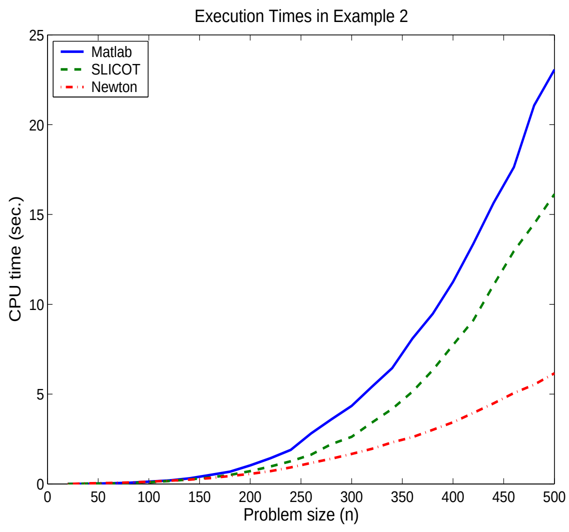


Figure 3: Relative errors and execution times of the Sylvester equation solvers applied to Example 2.

Example 3 [45, Example 13]. For the generalized Sylvester equation we choose

$$\begin{array}{rcll}
\hat{A} & = & \operatorname{diag}\left(1, a, a^2, \ldots, a^{n-1}\right), & a > 1, \\
\hat{B} & = & \operatorname{diag}\left(1, b^{-1}, b^{-2}, \ldots, b^{-(n-1)}\right), & b > 1, \\
\hat{D} & = & \operatorname{diag}\left(-1, -d^{-1}, -d^{-2}, \ldots, -d^{-(n-1)}\right), & d > 1, \\
\hat{E} & = & \operatorname{diag}\left(-1, -e, -e^2, \ldots, -e^{n-1}\right), & e > 1, \\
\hat{H} & = & vv^T, & \\
\hat{C} & = & -\hat{H}\hat{D} - \hat{H}\hat{B}, &
\end{array}$$

where the parameters a, b, d, and e regulate the eigenvalue distribution of the corresponding matrices. We then employ an equivalence transformation defined by T as in example 1 to transform the matrices of the equation into

$$\begin{array}{llll}
A = T^{-T}\hat{A}T^T, & B = T\hat{B}T^{-1}, & D = T\hat{D}T^{-1}, & E = T^{-T}\hat{E}T^T, \\
C = T^{-T}\hat{C}T^{-1}, & \text{and} & X = T^{-T}\hat{X}T^{-1}. &
\end{array}$$

In this example we set the parameters as $a = 1.001$, $b = 1.004$, $d = 1.002$, $e = 1.003$, and $s = 1.01$.

We compare the generalized Newton iteration for (1) as given in Algorithm (5) with the Bartels-Stewart and Hessenberg-Schur methods. Neither MATLAB nor SLICOT provide a solver for generalized Sylvester equations, so we use their standard Sylvester equation solvers, applied to (3). As all methods require the solution of linear systems of equations with D and E as coefficient matrices at the beginning (when using (3)) or the end (Algorithm (5)), an ill-conditioning of these matrices would affect all methods. Whether or not delaying this effect to the end is advantageous does not become obvious in this example as D and E are both well-conditioned with condition numbers less than 100.

Figure 4 compares the relative error in the solutions X for the three different methods and also shows the execution times needed. The relative errors for all three approaches are remarkably similar. For large problems, the generalized Newton iteration offers the lowest execution times.

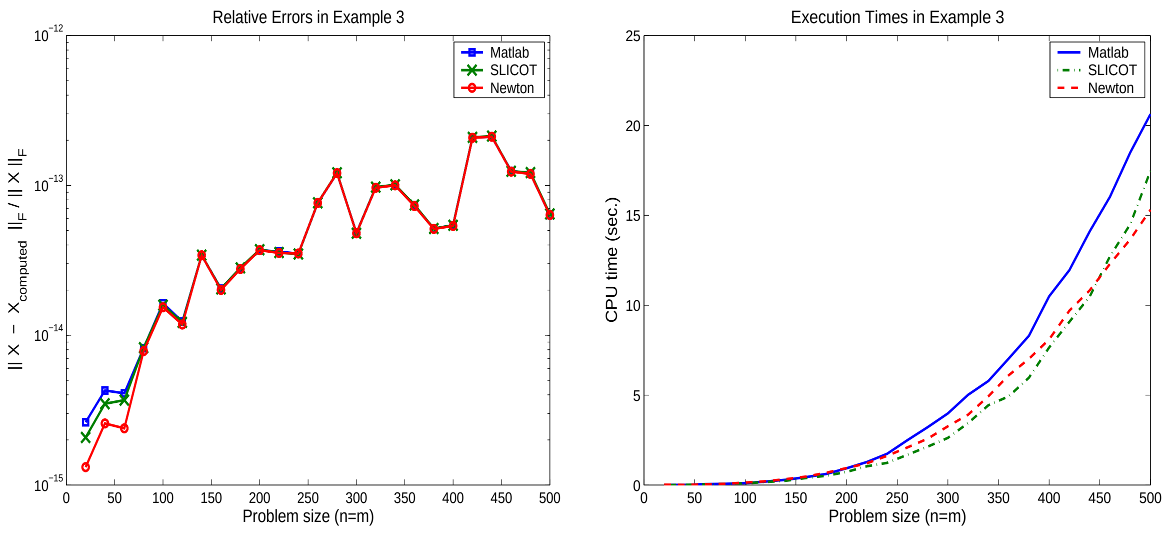


Figure 4: Relative errors and execution times of the generalized Sylvester equation solvers applied to Example 3.

5.2 Parallel implementation and performance

The iterative schemes that we have described in the previous sections are basically composed of traditional matrix computations such as matrix factorizations, solution of triangular linear systems, matrix inversion, and matrix products. All these operations can be efficiently performed employing parallel linear algebra libraries for distributed memory computers [12, 46]. The use of these libraries enhances the reliability and improves portability of the Sylvester equation solvers. The performance will depend on the efficiencies of the underlying serial and parallel computational linear algebra kernels and the communication routines.

Here we employ the parallel kernels in the ScaLAPACK library [12]. This is a freely available library that implements parallel versions of many of the routines in LAPACK [3], using the message-passing paradigm. ScaLAPACK is based on the PBLAS (a parallel version of the serial BLAS) for computation and BLACS for communication. The BLACS can be ported to any (serial and) parallel architecture with an implementation of the MPI [32].

Using the kernels in ScaLAPACK, we have implemented in Fortran-77 the following four computational routines for the solution of Sylvester equations:

– `pdgecsne`: The Newton iteration for the Sylvester equation.

– `pdgecsns`: A hybrid algorithm for the Sylvester equation that employs the Newton iteration until convergence of the Newton-Schulz iteration is guaranteed and, from then on, switches to the latter iterative scheme.

– `pdgecsha`: Halley's method for the Sylvester equation.

– `pdggcsne`: The generalized Newton iteration for the generalized Sylvester equation.

All the experiments presented in this section were performed on a cluster of 20 nodes using IEEE double-precision floating-point arithmetic ($\varepsilon \approx 2.2204 \times 10^{-16}$). Each node consists of an Intel Pentium Xeon processor at 2.4 GHz with 1 Gbyte of RAM. We employ a BLAS library specially tuned for the Pentium Xeon processor, that achieves around 3800 Mflops (millions of flops per second) for the matrix product (routine `DGEMM`) involving matrices of moderate dimensions [29]. The nodes are connected via a *Myrinet* multistage network; the communication library BLACS is based on an implementation of the MPI communication library specially developed and tuned for this network. The performance of the interconnection network was measured by a simple loop-back message transfer resulting in a latency of 18 μsec. and a bandwidth of 1.4 Gbit/sec. We made use of the LAPACK, PBLAS, and ScaLAPACK libraries whenever possible.

In the following, we report the parallel performance of our Sylvester equation solvers. In particular, as the performance is independent of the number of iterations, all the results shown here correspond to one iteration of the algorithms.

Our first experiment reports the execution time of the parallel routines for a system of dimension $n = 2500$. This is about the largest size we could solve on a single node of our cluster, considering the number of data matrices involved, the amount of workspace necessary for computations, and the size of the RAM per node. The left-hand plot in Figure 5 reports the execution time of one iteration of the parallel algorithms using n_p=1, 2, 4, 6, 8, and 10 nodes. The execution of the parallel algorithm on a single node is likely to require a higher time than that of a serial implementation of the algorithm (using, e.g., LAPACK and BLAS); however, at least for such large scale problems, we expect this overhead to be negligible compared to the overall execution time.

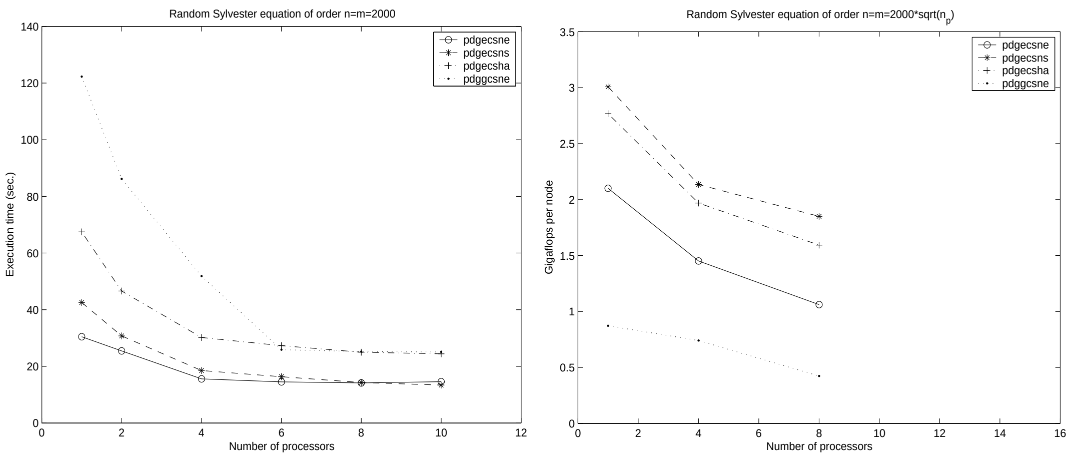


Figure 5: Performance (of one iteration) of the Sylvester equation solvers.

The figure shows reduced speed-ups when a small number of processors is employed. Thus, e.g., when n_p=2, speed-ups of 1.41, 1.55, 1.58, and 1.44 are obtained for routines `pdgecsne`, `pdgecsns`, `pdgecsha`, and `pdggcsne`, respectively. In all cases the efficiency decreases as n_p gets larger (as the system dimension is fixed, the problem size per node is reduced) so that using more than a few processors does not achieve a significant reduction in the execution time for such a small problem.

We next evaluate the scalability of the parallel algorithms when the problem size per node is constant. For that purpose, we fix the problem dimensions to $n/\sqrt{n_p} = 2000$, and report the Gflops per node. The right-hand plot in Figure 5 shows the Gflop rate per node of one iteration of the parallel routines. These results demonstrate the scalability of our parallel kernels, as there is only a minor decrease in the performance of the algorithms when n_p is increased while the problem dimension per node remains fixed.

6 Conclusions

We have proposed several variants of matrix sign function-based schemes for solving stable Sylvester equations. The resulting algorithms are based on rational iteration functions. Evaluation of these rational functions requires basic linear algebra computations which can be easily parallelized using a ScaLAPACK-style implementation. We have discussed how to efficiently implement the rational iterative schemes by exploiting the block-triangular matrix structure arising from applying the sign function method to solving Sylvester equations. Moreover, we have introduced a new iteration that delivers the solution of a Sylvester equation in factorized form while allowing significant savings in computation time and memory requirement in case the solution has low numerical rank. We have also discussed the application of the sign function method for a class of generalized Sylvester equations.

The experimental results confirm the expected behavior of the sign function-based solvers regarding numerical accuracy as well as efficiency and report a considerable degree of parallelism in the approach proposed here.

The observation that for stable matrices and matrix pencils with poor stability margin the norm scaling performs much better than determinantal scaling needs further justification by a thorough theoretical investigation.

Acknowledgments

We would like to express our gratitude to Vasile Sima of the National Institute for Research & Development in Informatics, Bucharest, Romania, for providing some helpful suggestions for improving the manuscript.

References

[1] J. Abels and P. Benner. CAREX – a collection of benchmark examples for continuous-time algebraic Riccati equations (version 2.0). SLICOT Working Note 1999-14, Nov. 1999. Available from `http://www.win.tue.nl/niconet/NIC2/reports.html`.

[2] R. Aldhaheri. Model order reduction via real Schur-form decomposition. *Internat. J. Control*, 53(3):709–716, 1991.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[4] A. Antoulas. *Lectures on the Approximation of Large-Scale Dynamical Systems.* SIAM Publications, Philadelphia, PA, to appear.

[5] Z. Bai and J. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In R. S. et al., editor, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 391–398. SIAM, Philadelphia, PA, 1993. *See also:* Tech. Report CSD-92-718, Computer Science Division, University of California, Berkeley, CA 94720.

[6] R. Bartels and G. Stewart. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Comm. ACM*, 15:820–826, 1972.

[7] A. N. Beavers and E. D. Denman. A new solution method for the Lyapunov matrix equations. *SIAM J. Appl. Math.*, 29:416–421, 1975.

[8] P. Benner. Factorized solution of Sylvester equations with applications in control. In *Proc. Intl. Symp. Math. Theory Networks and Syst. MTNS 2004*, `http://www.mtns2004.be`, 2004.

[9] P. Benner, J. Claver, and E. Quintana-Ortí. Parallel distributed solvers for large stable generalized Lyapunov equations. *Parallel Processing Letters*, 9(1):147–158, 1999.

[10] P. Benner and E. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. *Numer. Algorithms*, 20(1):75–100, 1999.

[11] P. Benner, E. Quintana-Ortí, and G. Quintana-Ortí. Numerical solution of discrete stable linear matrix equations on multicomputers. *Parallel Algorithms and Appl.*, 17(1):127–146, 2002.

[12] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, PA, 1997.

[13] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Linear Algebra Appl.*, 85:267–279, 1987.

[14] D. Calvetti and L. Reichel. Application of ADI iterative methods to the restoration of noisy images. *SIAM J. Matrix Anal. Appl.*, 17:165–186, 1996.

[15] T. Chan. Rank revealing QR factorizations. *Linear Algebra Appl.*, 88/89:67–82, 1987.

[16] C. Choi and A. Laub. Efficient matrix-valued algorithms for solving stiff Riccati differential equations. *IEEE Trans. Automat. Control*, 35:770–776, 1990.

[17] B. Datta. *Numerical Methods for Linear Control Systems Design and Analysis.* Elsevier Press, 2003.

[18] L. Dieci, M. Osborne, and R. Russell. A Riccati transformation method for solving linear BVPs. I: Theoretical aspects. *SIAM J. Numer. Anal.*, 25(5):1055–1073, 1988.

[19] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16:1–17, 1990.

[20] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, PA, 1991.

[21] W. Enright. Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations. *ACM Trans. Math. Softw.*, 4:127–136, 1978.

[22] M. Epton. Methods for the solution of $AXD - BXC = E$ and its application in the numerical solution of implicit ordinary differential equations. *BIT*, 20:341–345, 1980.

[23] K. Fernando and H. Nicholson. On a fundamental property of the cross-Gramian matrix. *IEEE Trans. Circuits Syst.*, CAS-31(5):504–505, 1984.

[24] Z. Gajić and M. Qureshi. *Lyapunov Matrix Equation in System Stability and Control.* Math. in Science and Engineering. Academic Press, San Diego, CA, 1995.

[25] J. Gardiner and A. Laub. Parallel algorithms for algebraic Riccati equations. *Internat. J. Control*, 54(6):1317–1333, 1991.

[26] J. Gardiner, A. Laub, J. Amato, and C. Moler. Solution of the Sylvester matrix equation $AXB + CXD = E$. *ACM Trans. Math. Software*, 18:223–231, 1992.

[27] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[28] G. H. Golub, S. Nash, and C. F. Van Loan. A Hessenberg–Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control*, AC-24:909–913, 1979.

[29] K. Goto and R. van de Geijn. On reducing TLB misses in matrix multiplication. FLAME Working Note 9, Department of Computer Sciences, The University of Texas at Austin, `http://www.cs.utexas.edu/users/flame`, 2002.

[30] L. Grasedyck. Existence of a low rank or H-matrix approximant to the solution of a Sylvester equation. *Numer. Lin. Alg. Appl.*, 11:371–389, 2004.

[31] L. Grasedyck and W. Hackbusch. A multigrid method to solve large scale Sylvester equations. Preprint 48, Max-Planck Institut für Mathematik in den Naturwissenschaften, Leipzig, Germany, 2004.

[32] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, Cambridge, MA, 1994.

[33] E. Halley. Methodus nova, accurata & facilis inveniendi radices aequationum quarumcumque generaliter, sine praevia reductione. *Philos. Trans. Roy. Soc. London*, 18:136–148, 1694.

[34] G. Henry and R. van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality. *SIAM J. Sci. Comput.*, 17:870–883, 1997.

[35] N. Higham. Computing the polar decomposition—with applications. *SIAM J. Sci. Statist. Comput.*, 7:1160–1174, 1986.

[36] W. Hoskins, D. Meek, and D. Walton. The numerical solution of the matrix equation $XA + AY = F$. *BIT*, 17:184–190, 1977.

[37] D. Hu and L. Reichel. Krylov-subspace methods for the Sylvester equation. *Linear Algebra Appl.*, 172:283–313, 1992.

[38] B. Kågström and P. Poromaa. Lapack-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs. *ACM Trans. Math. Software*, 22(1):78–103, 1996.

[39] C. Kenney and A. Laub. Rational iterative methods for the matrix sign function. *SIAM J. Matrix Anal. Appl.*, 12:273–291, 1991.

[40] C. Kenney and A. Laub. The matrix sign function. *IEEE Trans. Automat. Control*, 40(8):1330–1348, 1995.

[41] E. Quintana-Ortí, G. Quintana-Ortí, X. Sun, and R. van de Geijn. A note on parallel matrix inversion. *SIAM J. Sci. Comput.*, 22:1762–1771, 2001.

[42] J. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

[43] G. Schulz. Iterative Berechnung der reziproken Matrix. *Z. Angew. Math. Mech.*, 13:57–59, 1933. In German.

[44] V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, NY, 1996.

[45] M. Slowik, P. Benner, and V. Sima. Evaluation of the linear matrix equation solvers in SLICOT. SLICOT Working Note 2004–1, Sept. 2004. Available from `http://www.win.tue.nl/niconet/NIC2/reports.html`.

[46] R. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, 1997.

[47] E. Wachspress. Iterative solution of the Lyapunov matrix equation. *Appl. Math. Letters*, 107:87–90, 1988.