SLICOT Working Note 2014-1

On BLAS Level-3 Implementations of Common Solvers for (Quasi-) Triangular Generalized Lyapunov Equations [1]

Martin Köhler [2] Jens Saak[3]

September 2014

[1]This document presents research results obtained during the development of the SLICOT (Subroutine Library in Systems and Control Theory) software. The SLICOT Library and the related CACSD tools based on SLICOT were partially developed within the Numerics in Control Network (NICONET) funded by the European Community BRITE-EURAM III RTD Thematic Networks Programme (contract number BRRT-CT97-5040), see http://www.icm.tu-bs.de/NICONET. SLICOT can be used free of charge by academic users, see http://www.slicot.org, for solving analysis and synthesis problems of modern and robust control. This report is available from www.slicot.org/REPORTS/SLWN2014-1.pdf
[2]Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany; Email: koehlerm@mpi-magdeburg.mpg.de
[3]Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany; Email saak@mpi-magdeburg.mpg.de

Abstract

The solutions of Lyapunov and generalized Lyapunov equations are a key player in many applications in systems and control theory. Their stable numerical computation, when the full solution is sought, is considered solved since the seminal work of Bartels and Stewart [1]. A number of variants of their algorithm have been proposed, but none of them goes beyond BLAS level-2 style implementation. On modern computers, however, the formulation of BLAS level-3 type implementations is crucial to enable optimal usage of cache hierarchies and modern block scheduling methods based on directed acyclic graphs describing the interdependence of single block computations. Our contribution closes this gap by a transformation of the aforementioned level-2 variants to level-3 versions and a comparison on a standard multicore machine.

Keywords: Lyapunov Equation, Level-3 BLAS

1 Introduction

Lyapunov equations play an important role in systems and control theory. They are, e.g., a key ingredient in model order reduction, like Balanced Truncation [10], or part of the Newton Method for the Algebraic Riccati Equation [5]. Nowadays many practical situations require the solution of the generalized continuous-time Lyapunov equation

$$A^T X E + E^T X A = Y \tag{1}$$

or the generalized discrete-time Lyapunov equation, the so called generalized Stein equation

$$A^T X A - E^T X E = Y, \tag{2}$$

where A, E, X and Y are real $n \times n$ matrices. Furthermore we assume a symmetric right hand side Y such that the solution X is symmetric [11].

In order to recall the solvability conditions for both matrix equations we use that both equations are only a special case of the generalized Sylvester equation

$$M^T U N + O^T U P = Q. \tag{3}$$

Since (3) is linear in the entries of U, it can be reformulated as a linear system [13]:

$$\left(N^T \otimes M^T + P^T \otimes O^T\right) \mathrm{vec}(U) = \mathrm{vec}(Q), \tag{4}$$

where $\otimes$ denotes the Kronecker product of two matrices and $\mathrm{vec}(\cdot)$ the column-wise concatenation of an $n \times m$ matrix to a vector of length nm.

Using this formulation Chu [2] showed that Equation (3), and respectively Equation (4) are uniquely solvable if and only if the matrix pencils (M, O) and (P, N) are regular and have disjoint spectra. In the context of the generalized Lyapunov equation [11] these conditions simplify to: (1) has a unique solution if and only if $\lambda_i + \lambda_j \neq 0$ for any two eigenvalues λ_i and λ_j of (A, E). In the case of the generalized Stein equation we get: Equation (2) is uniquely solvable if and only if $\lambda_i \lambda_j \neq 1$ for any two eigenvalues λ_i and λ_j of (A, E) (under the assumption $0 \cdot \infty = 1$). As a direct consequence, if one of the matrices A and E is singular, the corresponding linear system (4) is singular, too.

Beside this theoretical background we briefly mention several algorithms for the solution of the generalized Lyapunov equation. The first and trivial idea is to solve Equation (1) through its reformulation to the Kronecker system (4). This obviously leads to very large dense linear systems. In avoidance of the resulting huge runtime complexity we have to use specialized Sylvester equation solvers that uses the special structure of the Kronecker product. The most popular variants are the Gardiner and Laub modification [3] of the Bartels-Stewart algorithm [1] and the generalized Schur algorithm by Kågström and Westin [7]. Focusing on modern computer architectures Kågström and Poromaa [6] developed a fast level-3 BLAS implementation of the generalized Schur algorithm on top of LAPACK. All these techniques do not care about the symmetric structure of the Lyapunov equation and so they can not guarantee the symmetry of the solution X from a numerical point of view.

A variant of the Bartels-Stewart algorithm which works directly on Equation (1) or (2) was presented by Penzl in 1997 [11]. The algorithm preserves the symmetry of the solution X, but in contrast to the previously mentioned Sylvester solvers, this algorithm only employs BLAS level-2 operations. This prevents an efficient usage of modern computer architectures and their cache hierarchy. Furthermore level-2 BLAS operations are not well suited for massive parallel accelerators like NVIDIA® CUDA devices or the Intel® Xeon® Phi. Regardless of this fact, this implementation is part of the SLICOT [12] software package and the backend behind the generalized Lyapunov equation solver in MATLAB® or GNU Octave.

In the following sections we discuss a reformulation of Penzl's variant of the Bartels-Stewart algorithm into a level-3 BLAS enabled version. In Section 5 we show the performance of our implementation and the comparability of the results.

2 The Bartels-Stewart Algorithm for the Generalized Lyapunov Equation

First, we recall Penzl's extension [11] to the Bartels-Stewart algorithm [1]. Based on this extension we point out where to modify the algorithm to end up with a level-3 BLAS implementation. The optimal block size is determined later in Section 5 by a set of benchmarks. We restrict our presentation to the generalized Lyapunov equation (1). The derivation of the algorithm for the generalized Stein equation works analogously by exchanging the roles of A and E at the corresponding positions.

Like in the original Bartels-Stewart algorithm we have to transform Equation (1) to an equivalent but more structured representation. Therefore, we determine the generalized Schur form (A_s, E_s) of the matrix pencil (A, E) by means of two orthogonal matrices Q and Z (e.g. using the QZ algorithm [9]):

$$\begin{aligned} A &= Q^T A_s Z, \\ E &= Q^T E_s Z. \end{aligned} \tag{5}$$

Now A_s is a quasi upper triangular matrix and E_s is an upper triangular one. Inserting the decomposition (5) in the generalized Lyapunov equation (1) and multiplying from the left with Z and from the right by Z^T leads to

$$A_s^T \underbrace{QXQ^T}_{X_s} E_s + E_s^T \underbrace{QXQ^T}_{X_s} A_s = \underbrace{ZYZ^T}_{Y_s}. \tag{6}$$

This equation is equivalent to our original Equation (1) and the solution X is restored by:

$$X = Q^T X_s Q. \tag{7}$$

We call Equation (6) (quasi-) triangular generalized Lyapunov equation. The triangular structure of the matrices A_s and E_s allows us to use a Bartels-Stewart like forward substitution scheme. Therefore, we partition Equation (6) into 1×1 or 2×2 blocks with respect to real eigenvalues or complex eigenvalue pairs on the diagonal of (A_s, E_s). This results in a p by p block representation of all matrices:

$$\begin{aligned} A_s &= \begin{pmatrix} A_{11} & \cdots & A_{1p} \\ & \ddots & \vdots \\ 0 & & A_{pp} \end{pmatrix}, \quad E_s = \begin{pmatrix} E_{11} & \cdots & E_{1p} \\ & \ddots & \vdots \\ 0 & & E_{pp} \end{pmatrix}, \\ X_s &= \begin{pmatrix} X_{11} & \cdots & X_{1p} \\ \vdots & \ddots & \vdots \\ X_{p1} & \cdots & X_{pp} \end{pmatrix}, \quad Y_s = \begin{pmatrix} Y_{11} & \cdots & Y_{1p} \\ \vdots & \ddots & \vdots \\ Y_{p1} & \cdots & Y_{pp} \end{pmatrix}. \end{aligned} \tag{8}$$

Because X_s is symmetric we only have to solve for $\frac{1}{2}p(p+1)$ blocks. The remaining blocks are known by symmetry. This ensures that the solution X_s will be symmetric in contrast to the solution using an arbitrary Sylvester equation solver mentioned in Section 1. The solution of (6) now gets equal to solving Sylvester equations

$$A_{kk}^T X_{kl} E_{ll} + E_{kk}^T X_{kl} A_{ll} = \hat{Y}_{kl} \tag{9}$$

with updated right hand sides $\hat{Y}_{kl}$:

$$\hat{Y}_{kl} = Y_{kl} - \sum_{\substack{i=1,j=1 \\ (i,j)\neq(k,l)}}^{k,l} \left(A_{ik}^T X_{ij} E_{jl} + E_{ik}^T X_{ij} A_{jl} \right) \tag{10}$$

for each block X_{kl}. Resembling the equivalent Kronecker product formulation in (4) Penzl [11] proposed to compute X_{kl} by solving

$$\left(E_{ll}^T \otimes A_{kk}^T + A_{ll}^T \otimes E_{kk}^T \right) \operatorname{vec}(X_{kl}) = \operatorname{vec}\left(\hat{Y}_{kl} \right), \tag{11}$$

which is a linear system of size 1, 2 or 4 depending on the size of A_{kk} and A_{ll}. We solve this for $k = 1, \ldots, p$ and $l = k, \ldots, p$ in a row-wise order. After one column is completed the corresponding row is updated by transposing the column into the row.

Before we analyze this algorithmic idea with respect to the influenced BLAS operations, we recall an efficient replacement for the Update (10) presented by Penzl [11] as well. We expand the update of $\hat{Y}_{kl}$ to:

$$\begin{aligned}\hat{Y}_{kl} =& Y_{kl} - \sum_{i=1}^{k} \left(A_{ik}^T \underbrace{\left(\sum_{j=1}^{l-1} X_{ij} E_{jl} \right)}_{X_{i,1:l-1} E_{1:l-1,l}} + E_{ik}^T \underbrace{\left(\sum_{j=1}^{l-1} X_{ij} A_{jl} \right)}_{X_{i,1:l-1} A_{1:l-1,l}} \right) \\ & - \sum_{i=1}^{k} \left(A_{ik}^T X_{il} E_{ll} + E_{ik}^T X_{il} A_{ll} \right).\end{aligned}$$

If we now rearrange this scheme in a way that we update $\hat{Y}_{kl}$ step by step whenever a block X_{ij} in the formulation above becomes known we can compute $\hat{Y}_{kl}$ in $2k - 1$ steps:

$$\begin{aligned} Y_{kl}^{(0)} &= Y_{kl}, \\ Y_{kl}^{(2i-1)} &= Y_{kl}^{(2i-2)} - A_{ik}^T X_{i,1:l-1} E_{1:l-1,l} - E_{ik}^T X_{i,1:l-1} A_{1:l-1,l}, && i = 1, \ldots, k \qquad (12) \\ Y_{kl}^{(2i)} &= Y_{kl}^{(2i-1)} - A_{ik}^T X_{il} E_{ll} - E_{ik}^T X_{il} A_{ll}, && i = 1, \ldots, k-1 \qquad (13) \\ \hat{Y}_{kl} &= Y_{kl}^{(2k-1)}. \end{aligned}$$

This two-step update scheme is applied as follows: First, we have to compute (13) directly after we have solved for X_{il}. The remaining update for $Y_{kl}^{(2i-1)}$ is performed right before we solve for the corresponding $\hat{Y}_{kl}$. This scheme reduces the complexity in contrast to the original formulation (10). The key observation here is that we have to compute the two matrix products $X_{i,1:l-1} E_{1:l-1,l}$ and $X_{i,1:l-1} A_{1:l-1,l}$ only once.

2.1 Derivation of the blocked Algorithm

A detailed look at the obtained algorithm shows us that the assumption of only having 1×1 or 2×2 blocks is no restriction for the updates of the right hand sides $\hat{Y}_{kl}$. The formula (10) and its efficiency improving reformulation (12) and (13) can be evaluated for an arbitrary block size N_B for the partitioning of the matrices (8) in the (quasi-) triangular Lyapunov equation (6). The only restriction to the block size N_B of a block (k, l) is that its size must be adjusted by ± 1 in the case we would split a complex eigenvalue pair.

The change from 1×1 or 2×2 blocks to a larger block size N_B in algorithm proposed by Penzl allows us to use matrix-matrix products in (12) and (13) instead of only dealing with matrix-vector products and vector-scalings. The usage of matrix-matrix products and other level-3 BLAS operations is a prerequisite for the optimal utilization of modern computer architectures. Switching from matrix-vector products to matrix-matrix products, however, only provides a performance gain if the blocks have sufficient size. The optimal block size N_B^{opt} strongly depends on the capabilities of the computer architecture, like cache size, multi threading, memory hierarchies or vectorization opportunities. For a practical implementation of the algorithm with other block sizes than 1×1 or 2×2 this means that the block size N_B is a free parameter and must not be restricted by a smart implementation of any part of the algorithm. The only allowed variation of the block size is to fit the quasi triangular structure of the matrix A or to fit the dimension when the algorithm works on the last row or column block.

Now the question is why do current implementations only use a block size of 1 or 2? The answer is obvious in view of the way how the internally arising Sylvester equations (9) are handled. If we allow an arbitrary block size N_B, this inner equation will result in a linear system of size $N_B^2 \times N_B^2$. Regarding the flop count of the LU decomposition [4] we need $\frac{2}{3}N_B^6$ flops to factorize the inner linear system and $2N_B^4$ flops to do the forward/backward solves.

Example 1. *We consider a generalized Lyapunov equation of dimension $n = 960$ and a block size of $N_B = 64$, which is used as good choice for many other algorithms in LAPACK. Partitioning the matrices into blocks of dimension $N_B = 64$ results in a 15×15 block matrix. Using the previously presented algorithm with this block size requires the solution of 120 inner Sylvester equations. Solving one of them costs around 45 GFlops. Without counting the cost for the update of the right hand side we need 5.4 TFlops only in the internal step. In contrast to this the algorithm with block size 1 needs only 7 GFlops to compute all the inner equations and the updates of the right hand side [11]. The ratio of 770 between the necessary flops of both block sizes cannot be compensated by parallelization or optimized hardware usage on the same computer.*

Example 1 shows us that using the same ideas like Penzl with a freely selectable block size N_B will make the algorithm slower for nearly every choice of N_B that will accelerate the right hand side updates. The only possibility is to find a more efficient replacement for the inner Sylvester equation solver. Before we discuss different approaches to resolve this problem in Section 3, we present a pseudo code implementation and the flop count for the forward substitution scheme depending on the block size N_B.

2.2 Outer Algorithm and Flop Count

We have already seen that the Bartels-Stewart algorithm for the Lyapunov equation (6) consists of two parts. The first one is the update scheme for the right hand side and the second one is the solution of the inner Sylvester equation. At the moment we regard the solution of the inner problem (9) as a black-box. In this way we denote the Bartels -Stewart algorithm as "outer Algorithm" without any details about the solution of the inner equations.

If we assume that our matrices (A_s, E_s), X_s and Y_s from (8) are partitioned in P_B blocks of size N_B, we are able to compute the solution blocks X_{kl} in the row-wise way shown in Algorithm 1. The algorithm only solves for the upper triangular part of X_s and constructs the lower triangular part by symmetry in Step 4.

The pseudo code in Algorithm 1 already shows how it is implemented in-place by overwriting Y_s by X_s. We only need N_B^2 floating point numbers (FPN) of additional memory to compute the matrix-matrix products in Steps 8, 9, 14 and 15. In the following paragraphs we derive the flop count for Algorithm 1. Furthermore, we show that the blocked variant is in the same asymptotic runtime class as the version presented by Penzl, if we use a moderate block size.

Without loss of generality we assume that the dimension n of the Lyapunov equation is a multiple of the block size N_B. Then we have $P_B = \frac{n}{N_B}$, and the Steps 8 and 9 are executed in every iteration except of the first one ($k = l = 1$), where both steps take the same number of flops. This leads to a total cost for these two steps of:

$$\sum_{k=1}^{P_B}\sum_{l=k}^{P_B}\left(8lN_B^3 - 4kN_B^3\right) \underbrace{- 4N_B^3}_{\text{first iteration}} = 2{N_B}^3{P_B}^3 + 2{N_B}^3{P_B}^2 - 4{N_B}^3. \tag{14}$$

The cost for solving one inner Sylvester equation (9) depends on the algorithm and the block size used, therefore, we model it by a function $F_{inner}(N_B)$. In Section 3 we derive the F_{inner} for all inner solvers.

Algorithm 1 Forward-Substitution for the generalized Lyapunov equation

Input: (A_s, E_s) and Y_s partitioned in P_B blocks of size N_B, like in (8)
Output: X_s solving the (quasi-) triangular Lyapunov equation (6)

1: $X_s := Y_s$
2: **for** $k = 1, \ldots, P_B$ **do**
3: **if** $k > 1$ **then**
4: $X_{k,1:k-1} := X_{1:k-1,k}^T$ {Copy the symmetric part.}
5: **end if**
6: **for** $l = k, \ldots, P_B$ **do**
7: **if** $l > 1$ **then**
8: $X_{k:l,l} := X_{k:l,l} - A_{k,k:l}^T X_{k,1:l-1} E_{1:l-1,l}$
9: $X_{k:l,l} := X_{k:l,l} - E_{k,k:l}^T X_{k,1:l-1} A_{1:l-1,l}$
10: **end if**
11: Solve $A_{k,k}^T X_* E_{l,l} + E_{k,k}^T X_* A_{l,l} = X_{k,l}$
12: $X_{k,l} := X_*$
13: **if** $k < l$ **then**
14: $X_{k+1:l,l} := X_{k+1:l,l} - A_{k,k+1:l}^T X_{k,l} E_{l,l}$
15: $X_{k+1:l,l} := X_{k+1:l,l} - E_{k,k+1:l}^T X_{k,l} A_{l,l}$
16: **end if**
17: **end for**
18: **end for**

Independent of the definition of F_{inner} Step 11 costs

$$\sum_{k=1}^{P_B} \sum_{l=k}^{P_B} F_{inner}(N_B) = \frac{1}{2}({P_B}^2 + P_B) F_{inner}(N_B) \tag{15}$$

flops. The remaining update Steps 14 and 15 are executed in every iteration except of the iterations where $l = k$. This leads to a cost of

$$\sum_{k=1}^{P_B} \sum_{l=k}^{P_B} \left(4lN_B^3 - 4kN_B^3 + 4N_B^3\right) \underbrace{-P_B \cdot 4N_B^3}_{\text{iterations } k=l} = \frac{2}{3}{N_B}^3{P_B}^3 + 2{N_B}^3{P_B}^2 - \frac{8}{3}{N_B}^3 P_B \tag{16}$$

flops for these two steps. By summing up Equations (14) to (16) we get an overall flop count of

$$\begin{aligned} &\frac{1}{2}(P_B^2 + P_B)\, F_{inner}(N_B) + \frac{8}{3}{N_B}^3{P_B}^3 + 4{N_B}^3{P_B}^2 - 4{N_B}^3 - \frac{8}{3}{N_B}^3 P_B \\ &\qquad = \frac{1}{2}(P_B^2 + P_B)\, F_{inner}(N_B) + N_B^3 \left(\frac{8}{3}P_B^3 + 4P_B^2 - \frac{8}{3}P_B - 4\right). \end{aligned} \tag{17}$$

In the case of $N_B = 1$, $P_B = n$ and $F_{inner}(1) = 4$ this yields the approximately $\frac{8}{3}n^3$ flops reported by Penzl [11]. Replacing P_B in Equation (17) by $\frac{n}{N_B}$ we get

$$\begin{aligned} &\frac{1}{2}\left(\frac{n^2}{N_B^2} + \frac{n}{N_B}\right) F_{inner}(N_B) + N_B^3 \left(\frac{8}{3}\frac{n^3}{N_B^3} + 4\frac{n^2}{N_B^2} - \frac{8}{3}\frac{n}{N_B} - 4\right) \\ &= \frac{1}{2}\left(\frac{n^2}{N_B^2} + \frac{n}{N_B}\right) F_{inner}(N_B) + \left(\frac{8}{3}n^3 + 4N_B n^2 - \frac{8}{3}N_B^2 n - 4N_B^3\right). \end{aligned} \tag{18}$$

We see that the asymptotic complexity class is not influenced by the selected block size N_B as long as N_B is of moderate size. Neglecting all lower order terms we get the same asymptotic flop count of $\frac{8}{3}n^3$ as long as the solution of the inner Sylvester equation does not get too expensive. A special case is if

we consider $N_B \to n$. Then we have $4N_B^3 \to 4N_B n^2$ and $\frac{8}{3}N_B^2 n \to \frac{8}{3}n^3$ which means that the flops performed by the outer algorithm will go down to 0 and the overall flop count is dominated by F_{inner}.

The flop count analysis showed us that the problem of developing a level-3 BLAS enabled solver for the Lyapunov equation reduces to finding a suitable Sylvester equation solver, whose flop count $F_{inner}(N_B)$ only increases moderately with an increasing block size N_B.

3 Solution of the inner Sylvester Equations

In the previous Section we showed that the forward substitution scheme presented by Penzl [11] can be directly used for a level-3 BLAS variant of the Bartels-Stewart algorithm. The only missing part is the proper treatment of the arising inner Sylvester equations

$$A_{kk}^T X_{kl} E_{ll} + E_{kk}^T X_{kl} A_{ll} = \hat{Y}_{kl},$$

or prototypically written

$$\hat{A}^T \hat{X} \hat{B} + \hat{C}^T \hat{X} \hat{D} = \hat{Y}, \tag{19}$$

efficiently without forming the corresponding Kronecker representation (4). For simplicity we assume that all matrices $(\hat{A}, \hat{C})$ are of dimension $\hat{n}$ and the matrices $(\hat{D}, \hat{B})$ are of dimension $\hat{m}$. The right hand side $\hat{Y}$ and the solution $\hat{X}$ are $\hat{n} \times \hat{m}$ matrices.

We state the following demands on the algorithm, that should be fulfilled to integrate it in the outer algorithm without an additional overhead:

- The transposition of the matrices $\hat{A}$ and $\hat{C}$ must be done implicitly without forming their transposes.
- The right hand side $\hat{Y}$ must be overwritten by the solution $\hat{X}$ in order to save memory to avoid additional copy operations in the outer algorithm.
- The matrices $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ are used read only so that the outer algorithm need not make backups of them.
- The algorithm should have at most a cubic flop count such that the solver does not dominate the overall flop count (18) for moderate block size N_B.

The outer algorithm guarantees that the pencils $(\hat{A}, \hat{C})$ and $(\hat{D}, \hat{B})$ are already in generalized Schur form which we assume for the rest of this section. This property helps us to modify common solvers for the generalized Sylvester equation (19) in the following subsections. For each of them we determine F_{inner} for the best and the worst case situation.

If the inner Sylvester equation arises from the generalized Stein equation we have to solve

$$A_{kk}^T X_{kl} A_{ll} - E_{kk}^T X_{kl} E_{ll} = \hat{Y}_{kl}$$

in every step. This requires only a few changes in the following considerations. The most important one is that the first part of the equation now includes two matrices that may have 2×2 diagonal blocks. In the context of the Sylvester equation (19) this can be interpreted as $\hat{B}$ and $\hat{D}$ changing their structural roles. This modification is straight forward and therefore omitted.

3.1 An Approach based on Gardiner and Laub

One idea to solve the inner Sylvester equation is a slightly modified version of the Bartels-Stewart approach by Gardiner et al. [3]. Our modification works straightforward by changing the original algorithm at the correct positions to satisfy our requirements and the structure of the Sylvester equation. Additionally, we describe how the involved linear systems are solved efficiently.

Our outer algorithm guarantees that

$$\hat{A}^T \hat{X} \hat{B} + \hat{C}^T \hat{X} \hat{D} = \hat{Y}$$

has the following structure

where the matrices $\hat{A}$ and $\hat{D}$ may have 2×2 diagonal blocks. This structure allows us to rewrite the k-th column of $\hat{Y}$ as

$$\hat{A}^T \sum_{l=1}^{k} \hat{B}_{lk} \hat{X}_{\cdot l} + \hat{C}^T \sum_{l=1}^{k+1} \hat{D}_{lk} \hat{X}_{\cdot l} = \hat{Y}_{\cdot k} \quad \text{for} \quad k = 1, \ldots, \hat{m}. \tag{20}$$

If we now assume that $\hat{D}_{k+1,k} = 0$, i.e., the diagonal block of $\hat{D}$ belongs to a real eigenvalue, we can compute the k-th column of the solution $\hat{X}$ by

$$\begin{aligned} \hat{B}_{kk} \hat{A}^T \hat{X}_{\cdot k} + \hat{A}^T \sum_{l=1}^{k-1} \hat{B}_{lk} \hat{X}_{\cdot l} + \hat{D}_{kk} \hat{C}^T \hat{X}_{\cdot k} + \hat{C}^T \sum_{l=1}^{k-1} \hat{D}_{lk} \hat{X}_{\cdot l} &= \hat{Y}_{\cdot k} \\ \left(\hat{B}_{kk} \hat{A} + \hat{D}_{kk} \hat{C} \right)^T \hat{X}_{\cdot k} &= \hat{Y}_{\cdot k} - \hat{A}^T \sum_{l=1}^{k-1} \hat{B}_{lk} \hat{X}_{\cdot l} - \hat{C}^T \sum_{l=1}^{k-1} \hat{D}_{lk} \hat{X}_{\cdot l}. \end{aligned} \tag{21}$$

This scheme allows us to solve successively for the columns of the solution $\hat{X}_{\cdot k}$, for $k = 1, \ldots, n$. In order to not violate our requirements for the solver we see that we need additional $\hat{n}^2$ FPN memory to assemble the matrix $\hat{B}_{kk} \hat{A} + \hat{D}_{kk} \hat{C}$. If both matrices $\hat{A}$ and $\hat{C}$ are upper triangular, i.e., they have no 2×2 blocks on the diagonal, the system is solved in $\hat{n}^2$ flops by forward/backward substitution. Otherwise one has to use a normal LU decomposition or other techniques. A more efficient way is to eliminate the sub-diagonal entries by applying Givens rotations to reduce the matrix to upper triangular form. Afterwards, we use the forward/backward substitution again. This idea needs additional $\hat{n}$ FPN memory to store the parameters of the Givens rotations in the worst case.

Now we consider the case, when $\hat{D}_{k+1,k} \neq 0$. This means the $(k+1)$-st column depends on the solution of column k and vice versa. In this case we have to solve for $\hat{X}_{\cdot k}$ and $\hat{X}_{\cdot k+1}$ in one step. Regarding Equation (20) for two consecutive columns k and $k+1$ of $\hat{Y}$ leads us to

$$\left(\hat{B}_{kk} \hat{A} + \hat{D}_{kk} \hat{C} \right)^T \hat{X}_{\cdot k} + \hat{D}_{k+1,k} \hat{C}^T \hat{X}_{\cdot k+1} = \hat{Y}_{\cdot k} - \hat{A}^T \sum_{l=1}^{k-1} \hat{B}_{lk} \hat{X}_{\cdot l} - \hat{C}^T \sum_{l=1}^{k-1} \hat{D}_{lk} \hat{X}_{\cdot l} = \bar{Y}_{\cdot k}$$

and

$$\begin{aligned} \left(\hat{B}_{k,k+1} \hat{A} + \hat{D}_{k,k+1} \hat{C} \right)^T \hat{X}_{\cdot k} + \left(\hat{B}_{k+1,k+1} \hat{A} + \hat{D}_{k+1,k+1} \hat{C} \right)^T \hat{X}_{\cdot k+1} \\ = \hat{Y}_{\cdot k+1} - \hat{A}^T \sum_{l=1}^{k-1} \hat{B}_{lk} \hat{X}_{\cdot l} - \hat{C}^T \sum_{l=1}^{k-1} \hat{D}_{lk} \hat{X}_{\cdot l} = \bar{Y}_{\cdot k+1}. \end{aligned}$$

These two equations are combined into a linear system of size $2\hat{n} \times 2\hat{n}$:

$$\begin{pmatrix} \hat{B}_{kk}\hat{A} + \hat{D}_{kk}\hat{C} & \hat{B}_{k,k+1}\hat{A} + \hat{D}_{k,k+1}\hat{C} \\ \hat{D}_{k+1,k}\hat{C} & \hat{B}_{k+1,k+1}\hat{A} + \hat{D}_{k+1,k+1}\hat{C} \end{pmatrix}^T \begin{pmatrix} \hat{X}_{\cdot k} \\ \hat{X}_{\cdot k+1} \end{pmatrix} = \begin{pmatrix} \bar{Y}_{\cdot k} \\ \bar{Y}_{\cdot k+1} \end{pmatrix} \tag{22}$$

to compute $\hat{X}_k$ and $\hat{X}_{k+1}$ at once. The assembly of the system matrix needs $4\hat{n}^2$ FPN extra memory. The system can either be solved by an LU approach, or alternatively by a more efficient but also more complicated scheme.

The efficient solution works in two steps. First we reorder the rows and the columns of the matrix from its natural order $(1, 2, \ldots, 2\hat{n}-1, 2\hat{n})$ to $(1, 1+\hat{n}, 2, 2+\hat{n}, \ldots, \hat{n}, 2\hat{n})$. This reordering converts the matrix from Equation (22) to block upper triangular form with blocks of size 2×2 or 4×4 on the diagonal [3]. After this reordering we solve the remaining system by block forward/backward substitution. The blocks on the diagonal are solved using an LU decomposition with complete pivoting in order to get a robust scheme. Beside the $4\hat{n}^2$ FPN memory for assembling the matrix we need additional $2\hat{n}$ FPN memory to concatenate two consecutive columns of $\hat{Y}$ into one column vector of length $2\hat{n}$. This is necessary because we can not guarantee that the two consecutive columns reside in a continuous memory location, i.e. the leading dimension of the matrix $\hat{Y}$ is not equal to the number of rows.

Another performance improving strategy is to compute $\hat{A}^T\hat{X}_{\cdot l}$ and $\hat{C}^T\hat{X}_{\cdot l}$ once and store them. This reduces the complexity of the right hand side updates from $\mathcal{O}(\hat{n}^4)$ to $\mathcal{O}(\hat{n}^3)$. Therefore, we reuse the memory from the matrix assembly to store these precomputed vectors. Combining the precomputation strategy with both cases of the inner linear system we get Algorithm 2 for the solution of the generalized Sylvester equation. We showed: the algorithm needs $4\hat{n}^2 + 2\hat{n}$ FPN extra memory in order to satisfy our requirements for the inner solver. Because of the fact that the outer algorithm already needs N_B^2 FPN memory for the computations in the right hand side update, we only need to allocate $3\hat{n}^2 + 2\hat{n}$ FPN extra memory.

The flop count of the algorithm strongly depends on the diagonal block structure of the matrices $\hat{A}$ and $\hat{D}$. Therefore we only determine the best case flop count which is achieved if $\hat{A}$ and $\hat{D}$ only have 1×1 blocks on their diagonal and the worst case situation when both matrices consist of only 2×2 blocks on the diagonal. We assume that $\hat{n} = \hat{m}$ which is almost always the case unless we have to resize a block to avoid the splitting of a complex eigenvalue pair in the outer Algorithm 1.

The best case situation involves the assembly and solution of $\hat{n}$ linear systems in Step 4 of Algorithm 2. One of them needs

$$\underbrace{3\hat{n}^2}_{\text{assemble}} + \underbrace{\hat{n}^2}_{\text{solve}} = 4\hat{n}^2$$

flops. The precomputation of the right hand side update costs $4\hat{n}^2$ flops. Each of the $(\hat{n}-k)$ updates per iteration now costs only $4\hat{n}$ flops. The summation over all iterations gives us the best case F_{inner} flop count for this algorithm:

$$\begin{aligned} F_{inner}^{(best)}(\hat{n}) &= \sum_{k=1}^{\hat{n}} \left(4\hat{n}^2 + 4\hat{n}^2 + \sum_{l=k+1}^{\hat{n}} 4\hat{n} \right) \\ &= 10\hat{n}^3 - 2\hat{n}^2. \end{aligned} \tag{23}$$

The worst case situation causes only $\frac{\hat{n}}{2}$ iterations because each iteration processes two columns at once. On the other hand, we have to solve a large linear system of size $2\hat{n} \times 2\hat{n}$ in Step 11 of Algorithm 2. Under the assumption that $\hat{A}$ and $\hat{D}$ only consist of 2×2 diagonal blocks the assembled matrix has $\frac{\hat{n}}{2}$ diagonal blocks of size 4×4. Factorizing and solving with one of those blocks costs 63 flops. Each block involves a matrix-vector product of size $4 \times (2\hat{n} - 4l)$ to update the right hand side in the substitution

Algorithm 2 Solution of the generalized Sylvester equation (19)

Input: $(A, C) \in \mathbb{R}^{n\times n}$ and $(D, B) \in \mathbb{R}^{m\times m}$ in real generalized Schur form, $Y \in \mathbb{R}^{n\times m}$.
Output: $X \in \mathbb{R}^{n\times m}$ solving $A^T XB + C^T XD = Y$

1: $k := 1$
2: **while** $k \leq n$ **do**
3: **if** $D_{k+1,k} = 0$ **then**
4: Solve $(B_{kk}A + D_{kk}C)^T X_{\cdot k} = Y_{\cdot k}$
5: $x_1 := A^T X_{\cdot k}, \quad x_2 := C^T X_{\cdot k}$
6: **for** $l = k+1, \ldots, m$ **do**
7: $Y_{\cdot l} := Y_{\cdot l} - B_{k,l}x_1 - D_{k,l}x_2$
8: **end for**
9: $k := k + 1$
10: **else**
11: Solve
$$\begin{pmatrix} B_{kk}A + D_{kk}C & B_{k,k+1}A + D_{k,k+1}C \\ D_{k+1,k}C & B_{k+1,k+1}A + D_{k+1,k+1}C \end{pmatrix}^T \begin{pmatrix} X_{\cdot k} \\ X_{\cdot k+1} \end{pmatrix} = \begin{pmatrix} Y_{\cdot k} \\ Y_{\cdot k+1} \end{pmatrix}$$
12: $x_1 := A^T X_{\cdot k}, \quad x_2 := C^T X_{\cdot k}$
13: $y_1 := A^T X_{\cdot k+1}, \quad y_2 := C^T X_{\cdot k+1}$
14: **for** $l = k+2, \ldots, m$ **do**
15: $Y_{\cdot l} := Y_{\cdot l} - B_{k,l}x_1 - D_{k,l}x_2$
16: $Y_{\cdot l} := Y_{\cdot l} - B_{k+1,l}y_1 - D_{k+1,l}y_2$
17: **end for**
18: $k := k + 2$
19: **end if**
20: **end while**

scheme. The sum over all blocks in the matrix leads to

$$\sum_{l=1}^{\frac{\hat{n}}{2}} (63 + 8\,(2\hat{n} - 4l)) = 4\hat{n}^2 + \frac{47}{2}\hat{n}$$

flops to solve one linear system in Step 11. Assembling the matrix costs additional $10\hat{n}^2$ flops. The precomputation of the update costs $8\hat{n}^2$ flops, where finally each update only needs $8\hat{n}$ flops. This gives us an overall worst case flop count of

$$\begin{aligned} F_{inner}^{(worst)}(\hat{n}) &= \sum_{k=1}^{\frac{\hat{n}}{2}} \left(10\hat{n}^2 + 4\hat{n}^2 + \frac{47}{2}\hat{n} + 8\hat{n}^2 + \sum_{l=2k+1}^{\hat{n}} 8\hat{n} \right) \\ &= 13\hat{n}^3 + \frac{31}{4}\hat{n}^2. \end{aligned} \tag{24}$$

In contrast to solving the generalized Sylvester equation (19) using its Kronecker representation we only need $13\hat{n}^3$ flops asymptotically in the worst case instead of $\frac{2}{3}\hat{n}^6$.

Inserting this with $\hat{n} = N_B$ into Equation (15) to count all inner Sylvester solves we get

$$\begin{aligned} \frac{1}{2}\left(\frac{n^2}{N_B^2} + \frac{n}{N_B}\right) F_{inner}^{(best)}(N_B) &= \frac{1}{2}\left(\frac{n^2}{N_B^2} + \frac{n}{N_B}\right)\left(10N_B^3 - 2N_B^2\right) \\ &= 5N_B^2 n + 5N_B n^2 - N_B n - n^2 \end{aligned}$$

flops. Together with the outer algorithm we get an overall best case flop count $F^{(best)}(n, N_B)$ of

$$F^{(best)}(n, N_B) = \frac{8}{3}n^3 - n^2 + 9n^2 N_B + \frac{7}{3}nN_B^2 - nN_B - 4N_B^3 \tag{25}$$

to solve a (quasi-) triangular generalized Lyapunov equation. The worst case situation leads to

$$\frac{1}{2}\left(\frac{n^2}{N_B^2}+\frac{n}{N_B}\right)F_{inner}^{(worst)}(N_B)=\frac{13}{2}N_B^2n+\frac{13}{2}N_Bn^2+\frac{31}{8}N_Bn+\frac{31}{8}n^2$$

flops to solve all inner Sylvester equations and finally to a worst case flop count $F^{(worst)}(n,N_B)$ of

$$F^{(worst)}(n,N_B)=\frac{8}{3}n^3+\frac{31}{8}n^2+\frac{21}{2}n^2N_B+\frac{23}{6}nN_B^2+\frac{31}{8}nN_B-4N_B^3 \tag{26}$$

for Algorithm 1. The flop counts (25) and (26) show us that as long as N_B is of moderate size the $\frac{8}{3}n^3$ flops of the outer algorithm will dominate the overall flop count. It is obvious that if we consider $N_B \to n$ we end up with the best or respectively the worst case flop count of the inner Sylvester equation solver.

3.2 An Approach based on Kågström and Westin

In the previous subsection we presented a first approach to handle the inner Sylvester equation (9). Beside the idea of Gardiner and Laub to extend the Bartels-Stewart algorithm Kågström and Westin [7] presented a forward/backward substitution for generalized Sylvester equations of the structure

$$\begin{aligned}\check{A}R-L\check{B} &= \check{C}\\ \check{D}R-L\check{E} &= \check{F}.\end{aligned} \tag{27}$$

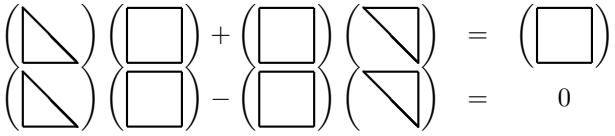
This corresponds to our Sylvester equation (19) by setting $\check{A}=\hat{A}^T$, $\check{B}=-\hat{D}$, $\check{C}=\hat{Y}$, $\check{D}=\hat{C}^T$, $\check{E}=\hat{B}$ and $\check{F}=0$ and solving $R=\hat{X}\hat{B}$ or $L=\hat{C}^T\hat{X}$ to retrieve the solution $\hat{X}$.

The algorithm to solve Equation (27) is available as `xTGSYL` and `xTGSY2` in LAPACK [6]. This implementation does not meet our requirements in order to get a fast inner solver in Algorithm 1. The LAPACK implementation requires that all input matrices are (quasi) upper triangular and non transposed. This violates our requirements because additional work is necessary and it does either modify our matrices or need a copy of all of them. Therefore we have to reformulate the generalized Schur algorithm [7] to work in-place with our matrices.

In our case, Equation (27) transforms to

$$\begin{aligned}\hat{A}^TR+L\hat{D} &= \hat{Y}\\ \hat{C}^TR-L\hat{B} &= 0 \quad = \hat{W},\end{aligned} \tag{28}$$

with $(\hat{A},\hat{C})\in\mathbb{R}^{\hat{n}\times\hat{n}}$, $(\hat{D},\hat{B})\in\mathbb{R}^{\hat{m}\times\hat{m}}$, $Y\in\mathbb{R}^{\hat{n}\times\hat{m}}$ and $W\in\mathbb{R}^{\hat{n}\times\hat{m}}$. The involved matrices result in the following structure:



All matrices are partitioned in $p\times q$ blocks of size 1×1 or 2×2 depending on the diagonal blocks in $\hat{A}$ and $\hat{D}$. Regarding this we can compute the blocks R_{11} and L_{11} directly by solving

$$\begin{aligned}\hat{A}_{11}^TR_{11}+L_{11}\hat{D}_{11} &= \hat{Y}_{11}\\ \hat{C}_{11}^TR_{11}-L_{11}\hat{B}_{11} &= \hat{W}_{11}\end{aligned}$$

which is either a 2×2, a 4×4 or an 8×8 system depending on the complex conjugate eigenvalue pairs in $(\hat{A},\hat{C})$ and $(\hat{D},\hat{B})$. Afterwards, we update the first column and the first row using R_{11} and L_{11} by

$$\begin{aligned}\hat{Y}_{2:n,1} &= \hat{Y}_{2:n,1}-\hat{A}_{1,2:n}^TR_{1,1}\\ \hat{W}_{2:n,1} &= \hat{W}_{2:n,1}-\hat{C}_{1,2:n}^TR_{1,1}\end{aligned}$$

and

$$\begin{aligned}\hat{Y}_{1,2:m} &= \hat{Y}_{1,2:m} - L_{1,1}\hat{D}_{1,2:m}\\ \hat{W}_{1,2:m} &= \hat{W}_{1,2:m} + L_{1,1}\hat{B}_{1,2:m}.\end{aligned}$$

This scheme is applied row by row to each block in R and L. In general this scheme looks as follows:

$$\begin{aligned}\hat{A}_{ii}^T R_{ij} + L_{ij}\hat{D}_{jj} &= \hat{Y}_{ij} - \sum_{k=1}^{i-1}\hat{A}_{ik}^T R_{kj} - \sum_{k=1}^{j-1} L_{ik}\hat{D}_{kj} &= \tilde{Y}_{ij}\\ \hat{C}_{ii}^T R_{ij} - L_{ij}\hat{B}_{jj} &= \hat{W}_{ij} - \sum_{k=1}^{i-1}\hat{C}_{ik}^T R_{kj} + \sum_{k=1}^{j-1} L_{ik}\hat{B}_{kj} &= \tilde{W}_{ij}\end{aligned} \tag{29}$$

for $i = 1, \ldots, p$ and $j = 1, \ldots, q$. The four sums to update the right hand sides can be computed at once by one matrix-matrix product per sum. The linear system in Equation (29) is rewritten to

$$\begin{pmatrix} I_{\hat{D}} \otimes \hat{A}_{ii}^T & \hat{D}_{jj}^T \otimes I_{\hat{A}} \\ I_{\hat{D}} \otimes \hat{C}_{ii}^T & -\hat{B}_{jj}^T \otimes I_{\hat{A}} \end{pmatrix} \begin{pmatrix} \mathrm{vec}(R_{ij}) \\ \mathrm{vec}(L_{ij}) \end{pmatrix} = \begin{pmatrix} \mathrm{vec}(\tilde{Y}_{ij}) \\ \mathrm{vec}(\tilde{W}_{ij}) \end{pmatrix}, \tag{30}$$

where $I_{\hat{A}}$ is an identity matrix of the dimension of $\hat{A}_{ii}$ and $I_{\hat{D}}$ is an identity matrix of the dimension of $\hat{D}_{jj}$. We need at most 64 FPN extra memory to assemble the system matrix if both $\hat{A}_{ii}$ and $\hat{D}_{jj}$ are 2×2 matrices. Additionally, we need 8 FPN extra memory for (R_{ij}, L_{ij}). Because our original Sylvester equation (27) has only one right hand side we need another $\hat{n}\hat{m}$ FPN extra memory to store the temporary second right hand side $\hat{W}$ which is set to 0 in the beginning. The memory for the solution (R, L) is not necessary because after we have solved for (R_{ij}, L_{ij}) the right hand side at block (i, j) is not needed any longer. In this way, the right hand side $(\hat{Y}, \hat{W})$ is overwritten by the solution (R, L).

After the computation of R and L we have to restore the solution $\hat{X}$ of the original Sylvester equation (19) by either solving

$$\hat{X} = \hat{C}^{-T} L \tag{31}$$

or

$$\hat{X} = R\hat{B}^{-1} \tag{32}$$

which both can be handled by BLAS's `xTRSM` routine because $\hat{B}$ and $\hat{C}$ are upper triangular matrices. In order to achieve the best quality of the result we choose between both variants using condition numbers of $\hat{B}$ and $\hat{C}$. Employing the upper triangular matrices we can cheaply compute the 2-norm condition number by

$$\mathrm{cond}_2(T) \approx \frac{\max(|\operatorname{diag}(T)|)}{\min(|\operatorname{diag}(T)|)}.$$

Algorithm 3 shows the overall procedure to solve our inner Sylvester equation using the generalized Schur algorithm of Kågström and Westin. Based on this algorithm we now determine the flop count for the best and worst case. Again, the best case is when both matrices $\hat{A}$ and $\hat{D}$ only have 1×1 blocks on their diagonal. The worst case is the other extreme situation where $\hat{A}$ and $\hat{D}$ consist of 2×2 diagonal blocks only. Caused by the usage inside the Lyapunov equation solver from Section 2 we assume $\hat{n} = \hat{m}$ again.

In the best case we now have $p = \hat{n}$ and $q = \hat{n}$. For each inner iteration we solve the the 2×2 linear systems in Step 4 employing 11 flops. The updates of both right hand sides in Step 6 and 7 cost $4(\hat{n} - i)$ flops and the updates in Steps 10 and 11 cost $4(\hat{n} - j)$ flops. Together with the final triangular solve we obtain

$$\begin{aligned}F_{inner}^{(best)}(\hat{n}) &= \hat{n}^3 + \sum_{i=1}^{\hat{n}}\sum_{j=1}^{\hat{n}} (11 + 4(\hat{n} - i) + 4(\hat{n} - j))\\ &= 5\hat{n}^3 + 7\hat{n}^2\end{aligned} \tag{33}$$

Algorithm 3 Solution of the generalized Sylvester equation (19)

Input: $(A, C) \in \mathbb{R}^{n\times n}$ and $(D, B) \in \mathbb{R}^{m\times m}$ in real generalized Schur form, $Y \in \mathbb{R}^{n\times m}$ partitioned in 1×1 and 2×2 blocks.

Output: $X \in \mathbb{R}^{n\times m}$ solving $A^T XB + C^T XB = Y$

1: $W := 0 \in \mathbb{R}^{n\times m}$
2: **for** $i = 1, \ldots, p$ **do**
3: **for** $j = 1, \ldots, q$ **do**
4: Solve $\begin{pmatrix} I_D \otimes A_{ii}^T & D_{jj}^T \otimes I_A \\ I_D \otimes C_{ii}^T & -B_{jj}^T \otimes I_A \end{pmatrix} \begin{pmatrix} \mathrm{vec}(R_{ij}) \\ \mathrm{vec}(L_{ij}) \end{pmatrix} = \begin{pmatrix} \mathrm{vec}(Y_{ij}) \\ \mathrm{vec}(W_{ij}) \end{pmatrix}$
5: **if** $i < p$ **then**
6: $Y_{i+1:p,j} := Y_{i+1:p,j} - A_{i,i+1:p}^T R_{i,j}$
7: $W_{i+1:p,j} := W_{i+1:p,j} - C_{i,i+1:p}^T R_{i,j}$
8: **end if**
9: **if** $j < q$ **then**
10: $Y_{i,j+1:q} := Y_{i,j+1:q} - L_{i,j} D_{j,j+1:q}$
11: $W_{i,j+1:q} := W_{i,j+1:q} + L_{i,j} B_{j,j+1:q}$
12: **end if**
13: **end for**
14: **end for**
15: **if** $\mathrm{cond}_2(B) < \mathrm{cond}_2(C)$ **then**
16: $X = RB^{-1}$
17: **else**
18: $X = C^{-T} L$
19: **end if**

as best case $F_{inner}^{(best)}$ flop count of the generalized Schur algorithm for this case.

In the worst case situation we have only 2×2 blocks in the matrices. That means $p = \frac{\hat{n}}{2}$ and $q = \frac{\hat{n}}{2}$. Therefore, one solution of the 8×8 system in Step (4) costs 415 flops. The updates in Steps 6 and 7 now cost $32(\frac{\hat{n}}{2} - i)$. Analogously, the updates in Steps 10 and 11 cost $32(\frac{\hat{n}}{2} - j)$. This leads to an overall worst case $F_{inner}^{(worst)}$ flop count of

$$\begin{aligned} F_{inner}^{(worst)}(\hat{n}) &= \hat{n}^3 + \sum_{i=1}^{\frac{\hat{n}}{2}} \sum_{j=1}^{\frac{\hat{n}}{2}} \left(415 + 32(\frac{\hat{n}}{2} - i) + 32(\frac{\hat{n}}{2} - j) \right) \\ &= 5\hat{n}^3 + \frac{383}{4}\hat{n}^2. \end{aligned} \tag{34}$$

In contrast to the solver based on the Gardiner-Laub approach from Subsection 3.1 the generalized Schur approach has the same constant in front of its highest order term for the best and the worst case flop count. That means that for larger dimensions the difference between the best and the worst case does not increase as fast as for the Gardiner-Laub approach. Furthermore, the generalized Schur algorithm is at least two times cheaper than our first approach.

Together with the Equation (15) and $\hat{n} = N_B$ all Sylvester equation solves need

$$\frac{1}{2}\left(\frac{n^2}{N_B^2} + \frac{n}{N_B}\right) F_{inner}^{(best)}(N_B) = \frac{5}{2}N_B^2 n + \frac{5}{2}N_B n^2 + \frac{7}{2}N_B n + \frac{7}{2}n^2$$

flops. Hence, we get an overall best case flop count $F^{(best)}$ of

$$F^{(best)}(n, N_B) = \frac{8}{3}n^3 + \frac{7}{2}n^2 + \frac{13}{2}N_B n^2 - \frac{1}{6}{N_B}^2 n + \frac{7}{2}N_B n - 4{N_B}^3 \tag{35}$$

to solve a (quasi-) triangular generalized Lyapunov equation (6). In the worst case situation, we need

$$\frac{1}{2}\left(\frac{n^2}{N_B^2}+\frac{n}{N_B}\right)F_{inner}^{(worst)}(N_B)=\frac{5}{2}N_B^2n+\frac{5}{2}N_Bn^2+\frac{383}{8}N_Bn+\frac{383}{8}n^2$$

flops to solver the inner Sylvester equations and get an overall flop count of

$$F^{(worst)}(n,N_B)=\frac{8}{3}n^3+\frac{383}{8}n^2+\frac{13}{2}N_Bn^2+\frac{383}{8}N_Bn-\frac{1}{6}{N_B}^2n-4{N_B}^3. \tag{36}$$

Like in the Gardiner and Laub approach the asymptotic flop count is $\frac{8}{3}n^3$ for moderate sizes of N_B. If we consider the case $N_B \to n$ we get the flop count of the inner Sylvester solver and the influence of the outer iteration vanishes.

3.3 Diagonal Blocks in the outer Algorithm

In case we have to solve for a block X_{kk} in Algorithm 1 the generalized Sylvester equation (19) simplifies to a generalized Lyapunov equation again. These blocks are solved with one of the two solvers presented, as well, but this may cause instabilities and lead to an inaccurate solution. From the introduction we know that for a given symmetric right hand side Y the solution X has to be symmetric, as well. However, in finite arithmetic the structure of both Sylvester solvers presented in Subsections 3.1 and 3.2 do not guarantee that the solution X_{kk} of

$$A_{kk}^TX_{kk}E_{kk}+E_{kk}^TX_{kk}A_{kk}=\hat{Y}_{kk}$$

satisfies

$$X_{kk}=X_{kk}^T. \tag{37}$$

The numerical results in Section 5 show that this error in fact increases with the block size N_B. In order to get a reliable outer algorithm, we have to ensure that Condition (37) even holds in floating point arithmetic for all diagonal blocks X_{kk} of X. The off-diagonal part of the solution is not affected by this problem because the outer algorithm constructs the lower triangular part of the solution by transposing the upper triangle as soon as the necessary blocks have been computed.

One can think of three different approaches to ensure a symmetric diagonal block in X:

1. The diagonal block X_{kk} is solved using Algorithm 2 or Algorithm 3 and the upper triangular part is copied to the lower triangular part. A possible problem of this idea is that we do not know if the upper or the lower triangular part is closer to the exact solution. This leads us to the next strategy.

2. The diagonal block X_{kk} is solved using Algorithm 2 or Algorithm 3 and we symmetrize the block by
$$\frac{1}{2}\left(X_{kk}+X_{kk}^T\right)\to X_{kk}. \tag{38}$$
The division by two is handled exactly by the IEEE floating point arithmetics in this way this operation will not disturb our solution.

3. The diagonal block X_{kk} is solved using a Lyapunov solver. In this way we use Algorithm 1 again with a smaller block size. This also decreases the overall flop count of the algorithm because instead of at least $\mathcal{O}(5N_B^3)$ flops for the Sylvester solvers the Lyapunov solver only needs $\mathcal{O}(\frac{8}{3}N_B^3)$ flops to solve for one diagonal block.

Due to the fact that the third idea leads to a recursion, and thus the same problem will reappear again, we focus on the second variant. The results in Section 5 show that this gives the most accurate results in average, as well.

	Gardiner and Laub	Kågström and Westin
Flop Count per Block		
best case	$10N_B^3 - 2N_B^2$	$5N_B^3 + 7N_B^2$
worst case	$13{N_B}^3 + \frac{31}{4}{N_B}^2$	$5N_B^3 + \frac{383}{4}N_B^2$
Flop Count Overall		
best case	$\frac{8}{3}n^3 + 9N_Bn^2 + \frac{7}{3}N_B^2n - 4N_B^3 - n^2 - N_Bn$	$\frac{8}{3}n^3 + \frac{7}{2}n^2 + \frac{13}{2}N_Bn^2 - \frac{1}{6}{N_B}^2n + \frac{7}{2}N_Bn - 4{N_B}^3$
worst case	$\frac{8}{3}n^3 + \frac{21}{2}N_Bn^2 + \frac{23}{6}N_B^2n + - 4N_B^3 + \frac{31}{8}\left(n^2 + N_Bn\right)$	$\frac{8}{3}n^3 + \frac{383}{8}(n^2 + N_Bn) - 4{N_B}^3 + \frac{13}{2}N_Bn^2 - \frac{1}{6}{N_B}^2n$
Additional Memory to the outer algorithm	$3N_B^2 + 2N_B$ FPN ($+N_B^2$ FPN from Algorithm 1)	72 FPN

Table 1: Comparison of the Sylvester solvers in the outer Algorithm

Summary of the inner Sylvester equation solvers. After we discussed two different approaches to solve the inner Sylvester equation (19) we summarize their implementation specific values and some advantages and disadvantages.

Table 1 shows the flop count and the extra memory which is necessary to solve a (quasi-) triangular generalized Lyapunov equation employing the two different inner solvers. We see that the Kågström-Westin approach yields a lower flop count and the worst and best case situations do not differ as much as for the Gardiner-Laub approach. Besides that the Kågström-Westin approach requires only N_B^2 FPN and some constant extra memory which is already covered by the extra memory requirements of the outer algorithm in contrast to the additional requirements of the Gardiner-Laub approach. The flop count and the extra memory requirements, however, do not give us any information how these variants will utilize modern computer architectures.

Other differences and caveats are:

- The Gardiner-Laub approach assembles relatively large linear systems. This causes additional memory transfers which may result in the deletion of required data from the cache. On the other hand, the involved operations work continuously on the data which is good from a CPU point of view. The updates of the right hand sides are, e.g, mostly vector additions that may reside completely in the CPU cache. Furthermore, the involved triangular solves exists in highly optimized variants in different BLAS implementations.
- The Kågström-Westin approach does not solves the generalized Sylvester equation (19) directly and needs an additional solve with $\hat{B}$ or $\hat{C}$, which is covered by BLAS as well. Although we are choosing the best conditioned one to solve with, this might lead to inaccurate solutions if both matrices are nearly singular. From the data locality point of view the updates of the right hand sides are irregularly structured in contrast to the Gardiner-Laub approach.

4 Memory Access Improvements

Beside the derivation of a level-3 BLAS enabled variant in the previous two sections we discuss an additional optimization which focuses on the memory access of the presented algorithm. On nearly all current computer architectures it is necessary to achieve an ordered and aligned memory access to get the maximum performance. On the lowest level this affects mostly the vector registers (SSE, AVX, AtliVec) of the CPU. In the case of the AVX instruction set of the x86 architecture the vector registers are able to deal with 4 double precision numbers at once. These registers are filled in two different ways. On the one hand, there exists an unaligned load, namely `VMOVUPD`, which transfers 4 double precision numbers

from an arbitrary memory location to an AVX registers. On the other hand, the aligned load, namely `VMOVAPD`, which moves 4 double precision numbers from a 32 byte aligned memory location to the AVX registers. The aligned load is much faster than its unaligned counterpart such that we have to use this for an optimal implementation.

In context of Algorithm 1 that means: if we want to achieve a maximum performance we have to guarantee that each block in the matrices A_s, E_s and X_s, as defined in Equation (8) is aligned in a way such that fast load and store operations are possible. If we stick to the case of AVX this results in an alignment of 32-byte (or 4 double precision numbers) for the start of each block in the matrices. This results in the following two conditions for the data layout:

1. The memory location of the first element of each column must be a multiple of the alignment. The easiest way to guarantee this property for arbitrary matrix sizes is to use a leading dimension n_{LD} which is maybe a bit larger than the matrix dimension n. The optimal leading dimension is determined via
$$n_{LD} := \left\lfloor \frac{(n + L - 1)}{L} \right\rfloor \cdot L,$$
where L denotes the alignment counted in floating point numbers of the desired precision and $\lfloor \cdot \rfloor$ is the floor operation. From the implementation point of view this does not influence anything because the used BLAS and LAPACK routines are able to handle arbitrary leading dimensions.

2. The block size N_B must be a multiple of the alignment such that each blocks starts with an aligned value and all subsequent elements in the block are aligned as well. In the case of an AVX enabled CPU and double precision arithmetics this will be 4, for older CPUs it is mostly 2.

The second condition is a bit problematic in our algorithm. In Subsection 2.1 we described that the actual block size N_B is adjusted by ± 1 to fit the eigenvalue structure of the matrix pencil. For example if we use a block size $N_B = 32$ and the $(32, 32)$ entry of the matrix A_s belongs to a complex eigenvalue pair we have to use either 31 or 33 as block size in this case. That means that all following blocks be will moved by one column and one row. The column shift does not disturb the alignment but the row shift does. The result is that even if we start with a well aligned matrix we might loose this property if the matrix pencil has at least one complex eigenvalue pair.

In order to handle this problem we have to permute the matrix pencil (A_s, E_s) such that each 2×2 diagonal block starts on an odd position. Using Givens rotations we can move the eigenvalues in (A_s, E_s) to any position on the diagonal [4]. Due to the fact that we do not want to introduce further restrictions to the block size we sort all complex eigenvalue pairs to the upper left on the diagonal. Because that are all 2×2 blocks we never get into trouble if the block size is a multiple of an alignment larger or equal to 2 floating point numbers. The block structure of the matrix A_s in Equation (8) then changes to

$$A_s := \begin{pmatrix} A_{1,1} & \dots & A_{1,k} & a_{1,k+1} & \dots & a_{1,p} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & A_{k,k} & a_{k,k+1} & \dots & a_{k,p} \\ & & & a_{k+1,k+1} & \dots & a_{k+1,p} \\ & & & & \ddots & \vdots \\ & & & & & a_{p,p} \end{pmatrix}, \tag{39}$$

where $A_{*,*}$ denotes a 2×2 block and $a_{*,*}$ a scalar value.

The structure in (39) can be easily computed via the QZ-algorithm implementation of LAPACK. By providing the additional *select* function one can choose which eigenvalues are moved to the upper left part.

In the numerical results we show how the adjustment of the leading dimension and the eigenvalue reordering influence the performance of Algorithm 1.

Block size N_B	1	2	4	8	16	24	32	40	48	56	64	72	80
SLICOT													
$n = 500$	0.56	-	-	-	-	-	-	-	-	-	-	-	-
$n = 1000$	4.79	-	-	-	-	-	-	-	-	-	-	-	-
$n = 2000$	44.55	-	-	-	-	-	-	-	-	-	-	-	-
Gardiner-Laub													
$n = 500$	0.63	0.57	0.35	0.17	0.14	**0.13**	0.14	0.14	0.15	0.16	0.17	0.18	0.20
$n = 1000$	5.17	4.82	2.99	1.12	0.81	0.74	**0.73**	0.74	0.76	0.79	0.83	0.86	0.91
$n = 2000$	46.32	43.23	26.55	7.89	5.25	4.59	4.39	**4.35**	4.36	4.45	4.55	4.67	4.80
Kågström-Westin													
$n = 500$	0.68	0.62	0.38	0.21	**0.18**	**0.18**	**0.18**	0.19	0.20	0.20	0.21	0.22	0.23
$n = 1000$	5.35	4.98	3.13	1.23	0.94	0.88	**0.87**	0.88	0.89	0.91	0.94	0.96	0.99
$n = 2000$	47.08	43.90	27.08	8.33	5.76	5.12	4.93	4.87	**4.84**	4.88	4.93	5.02	5.10

Table 2: Runtime in seconds for varying block sizes to solve a (quasi-) triangular generalized Lyapunov equation and different inner solvers, single-threaded BLAS.

5 Numerical Results

After we have seen (in Section 2) that the generalized Bartels-Stewart algorithm will work for arbitrary blocks sizes and how to handle the arising inner generalized Sylvester equations efficiently (in Section 3) we now test their performance and their accuracy. To this end, we use the following hard- and software setup. All tests are executed on a dual-socket Intel® Xeon® X5650 server with 12 cores and 48 GB main memory. The algorithms are written in Fortran 90, compiled with Intel® Fortran Compiler 13 and linked against Intel® MKL 11 as BLAS and LAPACK implementation. The reference results are obtained using the level-2 BLAS routines `SG03AY` from SLICOT 5.0 which is an improved version of the implementation by Penzl [11]. Additionally, we extend the SLICOT subroutine `SG03AD` which implements the overall procedure from Section 2 including the pre- and post-transformations and separation estimation. Therefore, we replaced all calls to `SG03AY` by our implementation and adjusted to memory requirements and the calling sequence to select the inner Sylvester solver and the block size N_B. All basic tests are done without the alignment improvements of Section 4, which are part of separate benchmarks at the end of this section.

As input data for the solver we use scalable matrix pencils. The first one is a random pencil to analyze the performance without focusing on a special eigenvalue structure of the matrix pencil. These matrices are generated via two subsequent calls of `DLARNV` from LAPACK. The initial seed for this subroutine is set to $(1, 1, 1, 1)$ and incremented during each call. These matrices do not have any specially tuned spectra so we see them as an average case for the algorithm. The second example is the artificial one already used by Penzl to check if the algorithm can handle ill-conditioned problems. As in [11, 3] we defined the matrices A and E as:

$$A = (2^{-t} - 1)I_n + \operatorname{diag}(1, 2, \ldots, n) + U_n$$
$$E = I_n + 2^{-t}U_n,$$

where I_n is the $n \times n$ identity and U_n is an $n \times n$ matrix with unit entries above the diagonal and all other entries zero. By increasing the parameter t we make the system more and more ill-conditioned. This example is similar to Example 4.3. from [8] but without transforming the matrices to generalized Schur form before. In all cases we set the true solution $X^{(true)}$ to a matrix with all unit entries and compute the right hand side Y appropriately. In the case where we only benchmark the Bartels-Stewart part we transform the pencil (A, E) to real generalized Schur form before we compute the right hand side. We skip extra benchmarks for the generalized Stein equation because they are similar from an algorithmic point of view and in practice we observed comparable results.

First, we want to obtain the optimal block size N_B for both variants of the inner Sylvester solver. Therefore, we run our implementation for three fixed dimensions $n \in (500, 1000, 2000)$ of the random

Block size N_B	1	2	4	8	16	24	32	40	48	56	64	72	80
SLICOT													
$n = 500$	0.56	-	-	-	-	-	-	-	-	-	-	-	-
$n = 1000$	4.93	-	-	-	-	-	-	-	-	-	-	-	-
$n = 2000$	41.98	-	-	-	-	-	-	-	-	-	-	-	-
Gardiner-Laub													
$n = 500$	0.65	0.57	0.35	0.17	0.13	**0.11**	0.12	0.12	0.12	0.13	0.14	0.15	0.17
$n = 1000$	5.21	4.86	2.66	0.94	0.57	**0.45**	0.46	0.48	0.48	0.52	0.56	0.58	0.63
$n = 2000$	43.81	40.04	18.47	6.11	3.16	2.10	2.13	2.23	**2.06**	2.25	2.39	2.41	2.61
Kågström-Westin													
$n = 500$	0.68	0.62	0.39	0.20	0.17	**0.15**	0.16	0.16	0.16	0.17	0.18	0.19	0.20
$n = 1000$	5.40	5.03	2.79	1.05	0.70	**0.59**	0.60	0.62	0.61	0.64	0.67	0.68	0.71
$n = 2000$	44.72	40.84	19.11	6.63	3.76	2.73	2.77	2.84	**2.63**	2.77	2.88	2.86	2.99

Table 3: Runtime in seconds for varying block sizes to solve a (quasi-) triangular generalized Lyapunov equation and different inner solvers, 12 threads.

matrices in generalized Schur form. We iterate over increasing block size N_B from 1 to 80 and compare to the SLICOT implementation. The symmetry of the diagonal blocks is guaranteed by using formula (38). Table 2 shows the runtime using the MKL in single thread mode. We observe that a block size of 24 or 32 leads to minimal runtime. Furthermore, we can accelerate the algorithm by a factor of 6.5 for $n = 1000$ or even 10.24 for $n = 2000$ in comparison to the SLICOT routine. Another observation is that, although we have shown in Section 3 that the Kågström-Westin approach needs less flops than the Gardiner-Laub approach, it is continuously a bit slower. The main reason for this might be the reuseability of data that is already in the CPU cache and the order in which the data is accessed. From Figure 1 we observe that even for larger block sizes the algorithm still works faster than the original implementation in SLICOT. That shows us that choosing a moderate block size larger than 8 accelerates the outer algorithm enough to compensate the increasing flop count of the inner Sylvester solver as long as we do not drive $N_B \to n$.

We run the same setup again with the MKL in multi-thread mode employing 12 threads. Table 3 shows us that here an optimal block size of 24 works as well but if the size of the matrix increase more, it may be useful to choose a larger block size like 48. Furthermore, we see that using a threaded BLAS implementation does not results in a mentionable speed up for the SLICOT implementation; it can even slow it down. In contrast to that our implementation using the Gardiner-Laub based inner solver gains a speed up of 10.95 for $n = 1000$, or even of 20.38 for $n = 2000$.

	Separation Est.					Forward Error Est.				
N_B	$t = 0$	$t = 10$	$t = 20$	$t = 30$	$t = 40$	$t = 0$	$t = 10$	$t = 20$	$t = 30$	$t = 40$
SLICOT										
1	2.00e+03	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.87e-12	1.17e-05	1.20e-02	1.23e+01	1.26e+04
Gardiner-Laub										
8	2.00e+03	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.87e-12	1.17e-05	1.20e-02	1.23e+01	1.26e+04
24	2.00e+03	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.87e-12	1.17e-05	1.20e-02	1.23e+01	1.26e+04
48	2.00e+03	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.87e-12	1.17e-05	1.20e-02	1.23e+01	1.26e+04
Kågström-Westin										
8	2.00e+03	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.87e-12	1.17e-05	1.20e-02	1.23e+01	1.26e+04
24	2.38e+00	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.42e-09	1.17e-05	1.20e-02	1.23e+01	1.26e+04
48	2.38e+00	4.77e-07	4.55e-13	4.34e-19	4.14e-25	2.42e-09	1.17e-05	1.20e-02	1.23e+01	1.26e+04

Table 4: Separation and Forward Error estimate for the generalized Lyapunov equation, Artificial Example $n = 1000$.

Beside the optimal block size we have to check that varying the block size does not influence the results. Therefore, we use the artificial example and check the relative residual $||A^T XE + E^T XA - Y||_F / ||Y||_F$, the relative forward error $||X^{true} - X^{computed}||/||X^{true}||_F$ and the separation and forward error estimates [11] computed by `SG03AD` from SLICOT. In extension to that we introduce two local error measures: First,

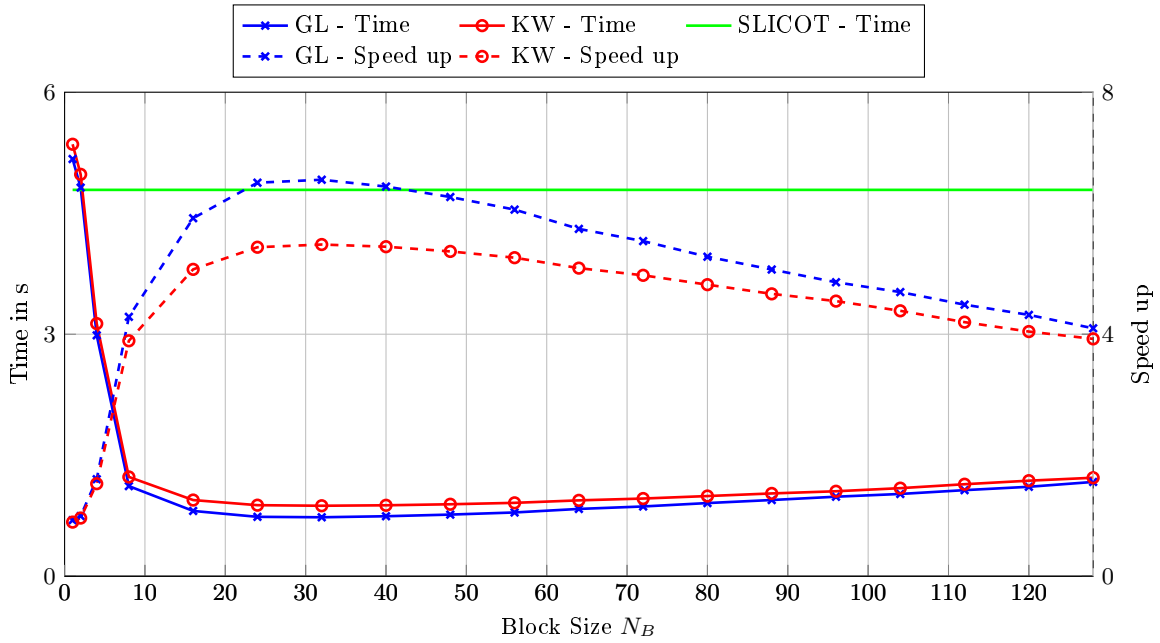


Figure 1: Runtime and speed up for a generalized Lyapunov equation, $n = 1000$, single thread. (GL = Gardiner and Laub, KW = Kågström and Westin)

we define the *Forbenius-normwise local relative residual*

$$||(A^T XE + E^T XA - Y)./Y||_F,$$

where "./" denotes the element-wise division and *Forbenius-normwise local forward error*

$$|| \left(X^{true} - X^{computed}\right) ./X^{true}||_F.$$

Second, we define the *maximal elementwise relative residual* and the *maximal elementwise relative forward error* as

$$\max_{i,j} \left(\frac{(A^T XE + E^T XA - Y)_{ij}}{Y_{ij}} \right)$$

and respectively

$$\max_{i,j} \left(\frac{X_{ij}^{true} - X_{ij}^{computed}}{X_{ij}^{true}} \right).$$

By construction our example guarantees that we do not divide by zero. We assume that the results from SLICOT represent the "true" values because computing the exact values is too expensive for large problems. Due to the fact that the matrices (A, E) are already upper triangular we do not have to compute their QZ decomposition before. The size of the artificial problem is fixed to $n = 1000$ and the parameter t varies from 0 to 40. The separation and the forward error estimate are computed by replacing `SG03AY` by our implementation in the driver routine `SG03AD` of SLICOT. The results in Table 4 show that even for the worstly conditioned problems $t = 40$ the SLICOT implementation and the Gardiner-Laub approach yield the same estimates for the separation and the forward error. The Kågström-Westin approach has some instabilities for an increasing block size. The reason behind this might be the additional solve to restore the solution of the inner Sylvester equations. The results for the relative residual and the computed forward error in Table 5 show the same behaviour. Switching to the

	Relative Residual					Relative Forward Error				
N_B	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$
SLICOT										
1	0.00	0.00	0.00	0.00	1.07e-15	3.54e-17	4.01e-17	4.17e-17	4.18e-17	5.97e-14
Gardiner-Laub										
8	0.00	0.00	0.00	0.00	5.24e-16	0.00	0.00	0.00	0.00	3.00e-14
24	0.00	0.00	0.00	0.00	4.15e-16	0.00	0.00	0.00	0.00	2.23e-14
48	0.00	0.00	0.00	0.00	4.17e-16	0.00	0.00	0.00	0.00	2.07e-14
Kågström-Westin										
8	1.62e-16	1.48e-16	1.44e-16	1.56e-16	5.14e-16	2.15e-14	2.03e-14	1.89e-14	2.14e-14	3.03e-14
24	1.78e-16	1.68e-16	1.72e-16	1.73e-16	4.10e-16	2.71e-14	2.69e-14	2.70e-14	2.72e-14	3.15e-14
48	2.02e-16	2.02e-16	2.05e-16	2.03e-16	3.83e-16	8.17e-14	8.14e-14	8.19e-14	8.07e-14	8.82e-14

Table 5: Relative Residual and Relative Forward Error for the generalized Lyapunov equation, Artificial Example $n = 1000$.

	Local Relative Residual					Local Relative Forward Error				
N_B	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$
SLICOT										
1	0.00	0.00	0.00	0.00	1.67e-12	3.54e-14	4.01e-14	4.17e-14	4.18e-14	6.25e-11
Gardiner-Laub										
8	0.00	0.00	0.00	0.00	8.86e-13	0.00	0.00	0.00	0.00	2.92e-11
24	0.00	0.00	0.00	0.00	7.67e-13	0.00	0.00	0.00	0.00	2.21e-11
48	0.00	0.00	0.00	0.00	7.70e-13	0.00	0.00	0.00	0.00	2.09e-11
Kågström-Westin										
8	1.49e-13	1.41e-13	1.39e-13	1.40e-13	8.98e-13	2.02e-11	1.90e-11	1.92e-11	1.93e-11	2.96e-11
24	1.95e-13	2.02e-13	1.96e-13	2.03e-13	7.58e-13	2.66e-11	2.63e-11	2.63e-11	2.65e-11	3.09e-11
48	3.50e-13	3.55e-13	3.65e-13	3.61e-13	8.46e-13	2.66e-11	2.63e-11	2.63e-11	2.65e-11	3.09e-11

Table 6: Frobenius-normwise Local Relative Residual and Local Relative Forward Error for the generalized Lyapunov equation, Artificial Example $n = 1000$.

local residual and forward error Tables 6 and 7 show that the Gardiner-Laub approach results in the most accurate results as well. Even in comparison to the SLICOT code we get a slightly smaller forward error. The Gardiner-Laub approach is closer or equal to the SLICOT results and computes a more accurate solution than the Kågström-Westin approach.

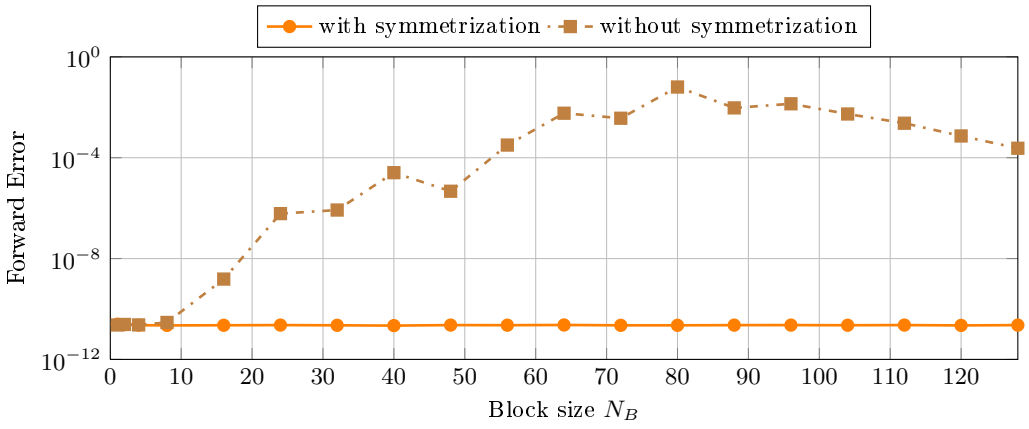


Figure 2: Forward error of the Gardiner-Laub approach with and without symmetrization the diagonal blocks.

All previously presented results used the symmetrization via Equation (38). Now we check the influence of this idea using the artificial example again. Figure 2 shows the relative forward error depending

	Local Relative Residual					Local Relative Forward Error				
N_B	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$	$t=0$	$t=10$	$t=20$	$t=30$	$t=40$
SLICOT										
1	0.00	0.00	0.00	0.00	6.26e-15	1.11e-16	1.11e-16	1.11e-16	1.11e-16	2.39e-12
Gardiner-Laub										
8	0.00	0.00	0.00	0.00	3.73e-15	0.00	0.00	0.00	0.00	8.23e-13
24	0.00	0.00	0.00	0.00	3.74e-15	0.00	0.00	0.00	0.00	5.32e-13
48	0.00	0.00	0.00	0.00	3.74e-15	0.00	0.00	0.00	0.00	4.32e-13
Kågström-Westin										
8	1.22e-15	1.20e-15	1.38e-15	1.35e-15	3.66e-15	5.34e-13	7.70e-13	6.43e-13	4.77e-13	9.47e-13
24	2.75e-15	3.80e-15	3.71e-15	4.08e-15	4.98e-15	4.54e-13	4.44e-13	3.91e-13	4.05e-13	4.27e-13
48	5.86e-15	8.97e-15	8.02e-15	1.15e-14	1.24e-14	1.71e-12	1.51e-12	1.73e-12	1.61e-12	1.65e-12

Table 7: Maximal elementwise Relative Residual and Relative Forward Error for the generalized Lyapunov equation, Artificial Example $n = 1000$.

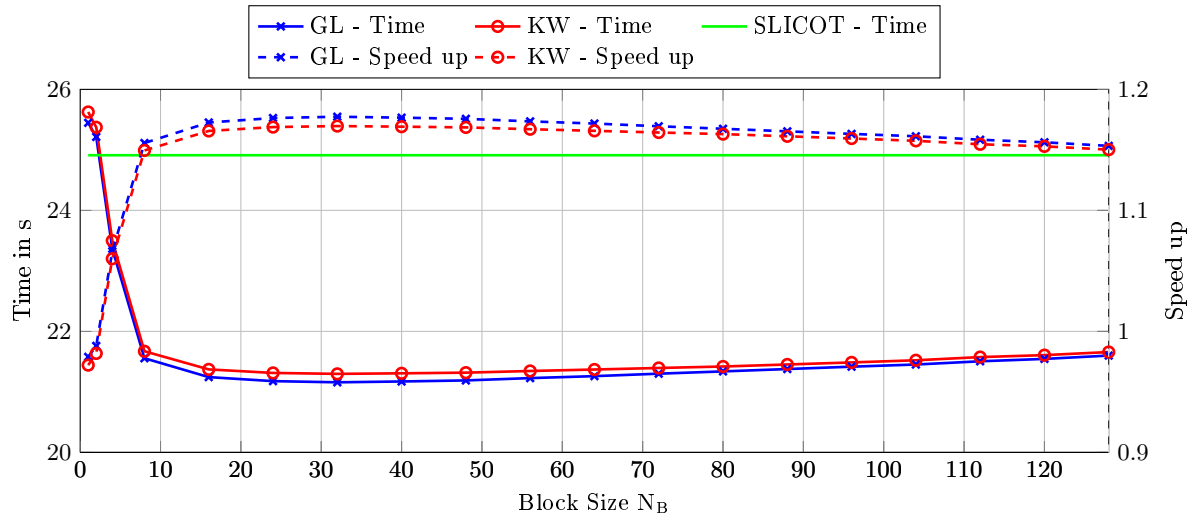


Figure 3: Runtime and speed up for a generalized Lyapunov equation including the QZ decomposition, $n = 1000$, single thread. (GL = Gardiner-Laub, KW = Kågström - Westin)

on the block size N_B for the Gardiner-Laub approach. We see that without the symmetrization the error will increase dramatically if we select a block size which accelerates the algorithm. From Table 4 we already know that employing the symmetrization formula the results have the same accuracy as the SLICOT implementation. Additional tests showed that the Kågström-Westin approach behaves similar and produces worse results without symmetrization, as well.

We perform the first benchmark again but with the unreduced random pencil. That means, before we can solve the Lyapunov equation we have to compute the QZ decomposition of the pencil (A, E), transform Y before and X afterward the actual solve. Figure 3 shows us the runtime and the speed up for the random pencil of dimension $n = 1000$ using a single-threaded BLAS. We can see that, even if we accelerate the Bartels-Stewart algorithm by a factor of nearly 6.5 in this case, the necessary precomputation will bring the overall speed up down to only 1.18. The reason behind this is that the computation of the generalized real Schur decomposition using the QZ algorithm is very expensive. Even if we use a multi-threaded BLAS this will not change very much because the design of the QZ algorithm, as it is implemented in LAPACK, prevents it from scaling on multi-core architectures. However, in situations, where the Lyapunov equation needs to be solved for different right hand sides and the Schur decomposition can be reused, the speedup will increase again.

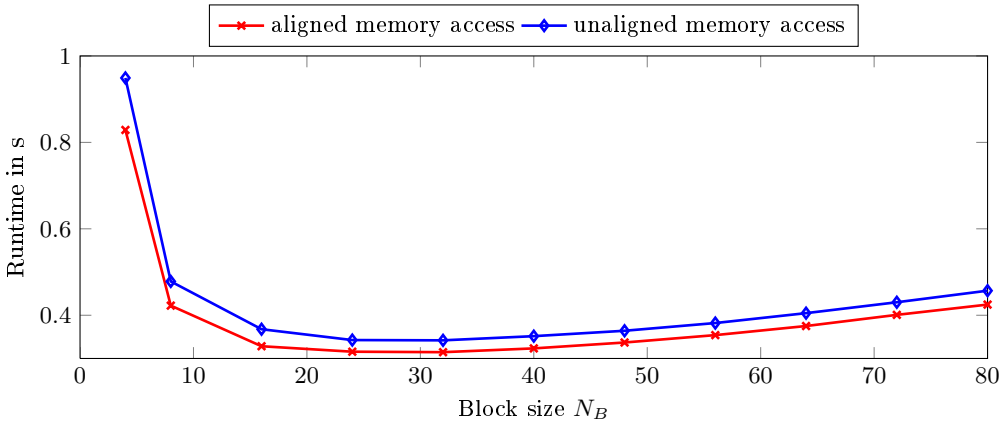


Figure 4: Influence of the alignment on the performance, $n = 1000$, average over 100 random pencils, single thread.

Block size N_B	4	8	16	24	32	40	48	56	64	72	80
Runtime unaligned in s	0.95	0.48	0.37	0.34	0.34	0.35	0.36	0.38	0.40	0.43	0.46
Runtime aligned in s	0.83	0.42	0.33	0.32	0.31	0.32	0.34	0.35	0.38	0.40	0.42
Speed up	1.15	1.13	1.12	1.09	1.09	1.09	1.08	1.08	1.08	1.07	1.08

Table 8: Runtime and Speed up for the aligned and unaligned memory access, $n = 1000$, single thread.

Influence of the alignment. After we have shown how our level-3 implementation compares to the existing SLICOT implementation we now want to show the influence of the data alignment. Therefore, we use another compute server equipped with two 8-core Intel® Xeon® E5-2690 CPUs and 32 GB RAM. The software setup stays the same as in the previous tests. The main difference is that the CPU has AVX registers, instead of only the SSE4.1 capabilities of the CPU used for the other results. The AVX registers require 32-byte aligned data to perform fast load and store operations. For our implementation this results in an alignment of 4 double precision floating point numbers. That means, the leading dimension n_{LD} for the matrix storage and the block size must be a multiple of 4. Furthermore, we only use the MKL in single thread mode to get rid of threading issues. The inner Sylvester equations are solved using the Gardiner and Laub strategy because this already turned out to be the best choice.

For the test we transform 100 randomly generated pencils of dimension $n = 1000$ to generalized Schur form and leave the eigenvalues unsorted for the first run and sort the eigenvalues as described in Section 4 for the second run. The results in Figure 4 and Table 8 show that the reordering of the eigenvalues and the resulting aligned memory can increase the performance of the level-3 BLAS implementation by 9% for the optimal block sizes $N_B = 24$ and $N_B = 32$. The 9% gain only measures the cases where the spectrum is already sorted or not and does not cover the additional time spent sorting the eigenvalues in the QZ algorithm. As already mentioned in the previous paragraph, the additional work in the QZ algorithm is compensated if we need to solve the Lyapunov equation for multiple right hand sides.

6 Conclusions

The numerical results show that using a block variant of the generalized Bartels-Stewart algorithm gains a good speed up in comparison to the legacy implementation in SLICOT. We have seen that using the Gardiner-Laub approach to solve the inner Sylvester equations results in the fastest variant with a comparable accuracy and reliability as the SLICOT implementation. Solving the inner Sylvester equations using the Kågström-Westin approach is slower, although the flop count analysis showed that it takes less operations than the Gardiner-Laub approach. Additionally, it can result in instabilities caused by the final solve in Algorithm 3. We observe that if we only accelerate the Bartels-Stewart part we can not gain a large speed up for the overall problem if we only have to solve with the equation once. The necessary QZ algorithm required to compute the structure exploited by the Bartels-Stewart algorithm stays the bottle neck of the overall procedure. As long as we already have the input data in generalized real Schur form, or we have to solve more than once with the same coefficients our reformulation is much faster. Accelerating the overall procedure and replacing the slow QZ algorithm by a multi-core aware variant is part of our current research.

Once we have a level-3 BLAS formulation of an algorithm we can easily derive accelerator based variants for GPUs or similar accelerators. Since accelerator devices are even more sensitive to unaligned memory accesses we have already shown that one can easily get an aligned memory access scheme for the whole procedure. Porting the algorithm to those devices will be addressed once the CPU version is finalized.

Beside the generalized Bartels-Stewart algorithm Penzl developed a generalization of Hammarling's Method [5, 11] as a level-2 BLAS implementation. Developing a level-3 implementation of this algorithm is another part of future research, although it appears to be a lot more challenging.

References

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

[2] E. K.-W. Chu, *The solution of the matrix equations AXB - CXD=E and (YA - DZ,YC - BZ)=(E,F)*, Linear Algebra Appl., 93 (1987), pp. 93 – 105.

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

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, third ed., 1996.

[5] S. Hammarling, *Newton's method for solving the algebraic Riccati equation*, NPL Report DITC 12/82, National Physical Laboratory, Teddington, Middlesex TW11 OLW, U.K., 1982.

[6] 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 (1996), pp. 78–103.

[7] B. Kågström and L. Westin, *Generalized Schur methods with condition estimators for solving the generalized Sylvester equation*, IEEE Trans. Automat. Control, 34 (1989), pp. 745–751.

[8] D. Kressner, V. Mehrmann, and T. Penzl, *CTLEX - a collection of benchmark examples for continuous-time Lyapunuv equations*, SLICOT Working Note 1999-6, June 1999. Available from www.slicot.org.

[9] C. B. Moler and G. W. Stewart, *An algorithm for generalized matrix eigenvalue problems*, SIAM J. Numer. Anal., 10 (1973), pp. 241–256.

[10] B. C. Moore, *Principal component analysis in linear systems: controllability, observability, and model reduction*, IEEE Trans. Automat. Control, AC-26 (1981), pp. 17–32.

[11] T. Penzl, *Numerical solution of generalized Lyapunov equations*, Adv. Comp. Math., 8 (1997), pp. 33–48.

[12] *SLICOT*, http://www.slicot.org.

[13] J. J. Sylvester, *The Collected Mathematical Papers of James Joseph Sylvester, Volume 4*, AMS Chelsea Publishing, 1973.

A Interface Description of SG03CD

Specification:

```
SUBROUTINE SG03CD( DICO, JOB, FACT, TRANS, UPLO, N, A, LDA, E, &
                 & LDE, Q, LDQ, Z, LDZ, X, LDX, SCALE, SEP, FERR, &
                 &  ALPHAR, ALPHAI, BETA, IWORK, DWORK, LDWORK, BWORK,&
                 &  INFO )
```

Purpose: To solve for X either the generalized continuous-time Lyapunov equation

$$\mathrm{op}(A)^T X\,\mathrm{op}(E) + \mathrm{op}(E)^T X\,\mathrm{op}(A) = sY \tag{40}$$

or the generalized discrete-time Lyapunov equation

$$\mathrm{op}(A)^T X\,\mathrm{op}(A) - \mathrm{op}(E)^T X\,\mathrm{op}(E) = sY, \tag{41}$$

where $\mathrm{op}(M)$ is either M or M^T for M = A, E and the right hand side Y is symmetric. A, E, Y, and the solution X are $N \times N$ matrices. s (SCALE) is an output scale factor, set to avoid overflow in X.

Estimates of the separation and the relative forward error norm are provided.

The function is based on SG03AD, SG03AX and SG03AY but in contrast to them this implementation uses a blocked level-3 BLAS variant of the Bartels-Stewart algorithm in order to achieve a higher performance on modern computer architectures. The blocksize NB of the inner algorithms (SG03CX and SG03CY) is fixed to 32. Additionally, the deprecated LAPACK routine DGEGS is replaced by DGGES.

Arguments:

Mode Parameters

DICO – CHARACTER*1
: Specifies which type of the equation is considered:
='C': Continuous-time equation (40)
='D': Discrete-time equation (41).

JOB – CHARACTER*1
: Specifies if the solution is to be computed and if the separation is to be estimated:
= 'X': Compute the solution only;
= 'S': Estimate the separation only;
= 'B': Compute the solution and estimate the separation.

FACT – CHARACTER*1
: Specifies whether the generalized real Schur factorization of the pencil $A - \lambda E$ is supplied on entry or not:
= 'N': Factorization is not supplied;
= 'F': Factorization is supplied.

TRANS – CHARACTER*1
: Specifies whether the transposed equation is to be solved or not:
= 'N': $\mathrm{op}(A) = A, \mathrm{op}(E) = E$;
= 'T': $\mathrm{op}(A) = A^T, \mathrm{op}(E) = E^T$.

UPLO – CHARACTER*1
Specifies whether the lower or the upper triangle of the array X is needed on input:
= 'L': Only the lower triangle is needed on input;
= 'U': Only the upper triangle is needed on input.

Input/Output Parameters

N – INTEGER, (input)
The order of the matrix A. N $\geq$ 0.

A – DOUBLE PRECISION array, dimension (LDA,N), (input/output)
On entry, if FACT = 'F', then the leading N-by-N upper Hessenberg part of this array must contain the generalized Schur factor A_s of the matrix A. A_s must be an upper quasitriangular matrix. The elements below the upper Hessenberg part of the array A must be zero. If FACT = 'N', then the leading N-by-N part of this array must contain the matrix A. On exit, the leading N-by-N part of this array contains the generalized Schur factor A_s of the matrix A. (A_s is an upper quasitriangular matrix.)

LDA – INTEGER
The leading dimension of the array A. LDA $\geq$ MAX(1,N).

E – DOUBLE PRECISION array, dimension (LDE,N), (input/output)
On entry, if FACT = 'F', then the leading N-by-N upper triangular part of this array must contain the generalized Schur factor E_s of the matrix E. The elements below the upper triangular part of the array E must be zero. If FACT = 'N', then the leading N-by-N part of this array must contain the coefficient matrix E of the equation. On exit, the leading N-by-N part of this array contains the generalized Schur factor E_s of the matrix E. (E_s is an upper triangular matrix.)

LDE – INTEGER
The leading dimension of the array E. LDE $\geq$ MAX(1,N).

Q – DOUBLE PRECISION array, dimension (LDQ,N), (input/output)
On entry, if FACT = 'F', then the leading N-by-N part of this array must contain the orthogonal matrix Q from the generalized Schur factorization. If FACT = 'N', Q need not be set on entry. On exit, the leading N-by-N part of this array contains the orthogonal matrix Q from the generalized Schur factorization.

LDQ – INTEGER
The leading dimension of the array Q. LDQ $\geq$ MAX(1,N).

Z – DOUBLE PRECISION array, dimension (LDZ,N), (input/output)
On entry, if FACT = 'F', then the leading N-by-N part of this array must contain the orthogonal matrix Z from the generalized Schur factorization. If FACT = 'N', Z need not be set on entry. On exit, the leading N-by-N part of this array contains the orthogonal matrix Z from the generalized Schur factorization.

LDZ – INTEGER
The leading dimension of the array Z. LDZ $\geq$ MAX(1,N).

X – DOUBLE PRECISION array, dimension (LDX,N), (input/output)
On entry, if JOB = 'B' or 'X', then the leading N-by-N part of this array must contain the right hand side matrix Y of the equation. Either the lower or the upper triangular part of this array is needed (see mode parameter UPLO). If JOB = 'S', X is not referenced. On exit, if JOB = 'B' or 'X', and INFO = 0 then the leading N-by-N part of this array contains the solution matrix X of the equation.

LDX – INTEGER
The leading dimension of the array Z. LDX $\geq$ MAX(1,N).

SCALE – DOUBLE PRECISION (output)
The scale factor set to avoid overflow in X. ($0 <$ SCALE ≤ 1)

SEP – DOUBLE PRECISION (output)
If JOB = 'S' or JOB = 'B', and INFO = 0 then SEP contains an estimate of the separation of the Lyapunov operator.

FERR – DOUBLE PRECISION (output)
If JOB = 'B', and INFO = 0 then FERR contains an estimated forward error bound for the solution X. If $X^{(true)}$ is the true solution, FERR estimates the relative error in the computed solution, measured in the Frobenius norm: $\frac{||X-X^{(true)}||_F}{||X^{(true)}||_F}$.

ALPHAR – DOUBLE PRECISION array, dimension(N) (output)

ALPHAI – DOUBLE PRECISION array, dimension(N) (output)

BETA – DOUBLE PRECISION array, dimension(N) (output)
If FACT = 'N' and INFO = 0, 3, or 4, then $\frac{\texttt{ALPHAR(j)}+\texttt{ALPHAI(j)}\imath}{\texttt{BETA(j)}}$, $j = 1, \ldots, N$, are the eigenvalues of the matrix pencil $A - \lambda E$. If FACT = 'F', ALPHAR, ALPHAI, and BETA are not referenced.

Workspace

IWORK – INTEGER array, dimension(IDWORK)
If JOB = 'X' then IDWORK = 66 or otherwise IDWORK = N^2 + 66

DWORK – DOUBLE PRECISION array, dimension(LDWORK)
On exit, if INFO = 0 and LDWORK=-1, DWORK(1) returns the optimal value of LDWORK.

LDWORK – INTEGER
The length of the array DWORK. The following table contains the minimal work space requirements depending on the choice of JOB and FACT.

JOB	FACT	LDWORK
'X'	'F'	MAX(1,N^2)
'X'	'N'	MAX(1,N^2,4488,8N+16)
'B', 'S'	'F'	MAX(1,2N^2,4488,8N+16)
'B', 'S'	'N'	MAX(1,2N^2,4488,8N+16)

If LDWORK = -1 on entry a workspace query is performed and the necessary workspace is returned in DWORK(1). No computations are done in this case.

BWORK – LOGICAL array, dimension(N)
BWORK is not referenced if FACT = 'F'.

Error Indicator

INFO – INTEGER

= 0 – successful exit;

< 0 – if INFO = -i, the i-th argument had an illegal value;

= 1 – FACT = 'F' and the matrix contained in the upper Hessenberg part of the array A is not in upper quasitriangular form;

= 2 – FACT = 'N' and the pencil $A - \lambda E$ cannot be reduced to generalized Schur form: LAPACK routine DGGES has failed to converge;

= 3 – DICO = 'D' and the pencil $A - \lambda E$ has a pair of reciprocal eigenvalues. That is, $\lambda_i = \lambda_j^{-1}$ for some i and j, where λ_i and λ_j are eigenvalues of $A - \lambda E$. Hence, equation (41) is singular; the computed solution is incorrect;

= 4 – DICO = 'C' and the pencil $A - \lambda E$ has a degenerate pair of eigenvalues. That is, $\lambda_i = -\lambda_j$ for some i and j, where λ_i and λ_j are eigenvalues of $A - \lambda E$. Hence, equation (40) is singular; the solution is not correct.