SIAM J. SCI. COMPUT.
Vol. 42, No. 2, pp. A957–A996

A NUMERICAL COMPARISON OF DIFFERENT SOLVERS FOR LARGE-SCALE, CONTINUOUS-TIME ALGEBRAIC RICCATI EQUATIONS AND LQR PROBLEMS*

PETER BENNER†, ZVONIMIR BUJANOVIĆ‡, PATRICK KÜRSCHNER§, AND JENS SAAK†

Abstract. In this paper, we discuss numerical methods for solving large-scale continuous-time algebraic Riccati equations. These methods have been the focus of intensive research in recent years, and significant progress has been made in both the theoretical understanding and efficient implementation of various competing algorithms. There are several goals of this manuscript. The first is to gather in one place an overview of different approaches for solving large-scale Riccati equations, and to point to the recent advances in each of them. The second goal is to analyze and compare the main computational ingredients of these algorithms and to detect their strong points and their potential bottlenecks. Finally, we want to compare the effective implementations of all methods on a set of relevant benchmark examples, giving an indication of their relative performance.

Key words. algebraic Riccati equation, Lyapunov equation, alternating direction implicit, rational Krylov subspaces, Newton's method

AMS subject classifications. 15A24, 65F45, 65F55, 93B52

DOI. 10.1137/18M1220960

1. Introduction.

Let $A,\ M \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{p\times n}$, $B \in \mathbb{R}^{n\times m}$ be given matrices. Assuming that A, M are sparse, M is nonsingular, and $p, m \ll n$, we consider large-scale, generalized, continuous-time, algebraic Riccati equations (GCAREs)

$$\mathcal{R}(X) = A^*XM + M^*XA - M^*XBB^*XM + C^*C = 0. \tag{1.1}$$

Our goal is the fast and efficient computation of a low-rank approximation of a solution matrix $X \in \mathbb{R}^{n\times n}$.

For $M = I_n$, (1.1) will be referred to as the standard continuous-time, algebraic Riccati equation (CARE). If $B = 0$, (1.1) reduces to a generalized, continuous-time, algebraic Lyapunov equation (GCALE). GCAREs appear in various areas related to control theory—for instance, linear-quadratic optimal regulator (LQR) problems [74, 82], H_2 and H_∞ controller design, nonlinear controller design via state-dependent Riccati equations [38], and balancing-related model reduction [4, 19, 66]. Solving differential Riccati equations by implicit integration schemes [29, 39, 45, 75] can also lead to GCAREs.

1.1. Preliminaries, assumptions, motivation, and goals.

Because of its nonlinear nature, (1.1) can have several solutions. We exclusively restrict ourselves to

*Submitted to the journal's Methods and Algorithms for Scientific Computing section October 31, 2018; accepted for publication (in revised form) February 3, 2020; published electronically April 7, 2020.

https://doi.org/10.1137/18M1220960

†Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg D-39106, Germany (benner@mpi-magdeburg.mpg.de, saak@mpi-magdeburg.mpg.de).

‡Faculty of Science, Department of Mathematics, University of Zagreb, Zagreb 10000, Croatia (zbujanov@math.hr).

§Group Science, Engineering and Technology, KU Leuven Kulak, 8500 Kortrijk, Belgium and Department of Electrical Engineering ESAT/STADIUS, KU Leuven, 3001 Leuven, Belgium (patrick.kurschner@kuleuven.be).

the usual situation where a stabilizing solution is sought; i.e., our goal is to find $X = X^* \geq 0$ such that the spectrum of the closed loop matrix fulfills $\Lambda(A - BB^*XM, M) \subset \mathbb{C}_-$. The stabilizing solution exists and is unique [35, 74] when (A, M, B) is stabilizable (i.e., $\operatorname{rank}[A - zM,\ B] = n$ for each value of z in the closed right half plane), and (A, M, C) is detectable ((A^*, M^*, C^*) stabilizable). These conditions are generically fulfilled and are assumed to hold in the remainder. We also assume that the singular values of X decay rapidly towards machine precision. This enables us to numerically deal with large-scale GCAREs (1.1) by computing low-rank approximations $X \approx X^{\mathrm{lr}} = ZDZ^*$ with $Z \in \mathbb{R}^{n\times r}$, $D = D^* \in \mathbb{R}^{r\times r}$, $r \ll n$. For GCALEs, the singular value decay has been investigated, e.g., in [5, 8, 25, 56, 87, 89, 97]. The low-rank approximability of solutions of GCAREs is less well understood; see, e.g., [16, 71, 88] for some results. In particular, a rapid singular decay of the solution is often present when $\Lambda(A, M) \subset \mathbb{C}_-$ and $p, m \ll n$, i.e., C^*C, BB^* are of low rank, which we assume as well and which is common in large-scale applications. The computation of low-rank factors Z, D is the common backbone shared by all the discussed algorithms in this paper. An efficient execution of these methods will also hinge on our assumption $p, m \ll n$.

The focus of this study is to compare the most prominent algorithms for large-scale GCAREs. The comparison is carried out with respect to the key stages of the algorithms which consume the most numerical effort. In particular, since all the considered methods are of an iterative nature often requiring different numbers of steps, we will analyze a single iteration step of each method. The main work-intensive stages of the methods that we are especially concerned with are as follows:

- The numerical solution of the large-scale, real- or complex-valued linear systems of equations, usually of the form
$$(A + \alpha M)V = N, \quad V, N \in \mathbb{C}^{n\times s},$$
with A, M from (1.1). We assume that we are able to solve these by sparse-direct or iterative solvers at an approximately linear complexity. Here, we focus on sparse-direct solvers. Employing iterative solvers deserves further investigations, but this would be well beyond the scope of this paper due to the large number of possible combinations of iterative schemes and preconditioners that could be used for this purpose. The number of columns s in the right-hand side is of special interest. We employ the simplifying assumption that the numerical effort to solve a linear system scales linearly with s.
- The construction of rectangular matrices with orthonormal columns spanning a basis for certain subspaces is a numerical task arising in methods that use a Galerkin projection framework. This is typically achieved by a stable implementation of a Gram–Schmidt, process, e.g., (modified) Gram–Schmidt with iterative refinement. In the numerical costs for the orthogonalization we will include only the orthogonalization carried out for a Galerkin projection regarding (1.1), and not orthogonalization stages arising in other parts of the algorithms, such as the computation of shifts or the residual norm.
- A Galerkin projection naturally leads to GCAREs (1.1) defined by dense matrices of smaller dimension $\ell \ll n$. Stable, numerical algorithms for small, dense GCAREs can be found, e.g., in [15, Chapter 1] and the references therein. If not stated otherwise, the MATLAB routine `care` is used, whose costs are estimated as $\mathcal{O}(\ell^3)$.
- The majority of the considered methods relies on certain shift parameters, which are important for rapid convergence. The problem of generating and

selecting high quality shift parameters might easily fill a survey article itself and is, therefore, beyond the scope of this study. We refer to the relevant literature [17, 24, 48, 49, 72, 101, 103] and provide only basic, necessary information where appropriate. For each particular GCARE solution method, we will employ the shift strategy providing the best results. With some exceptions, these are usually the more recent adaptive generation strategies.

- All the considered algorithms are of an iterative nature and, hence, require appropriate stopping criteria. For consistency, we terminate all algorithms when

$$\|\mathcal{R}(\tilde{X})\|_2 \leq \tau_{\mathcal{R}} \|C^*C\|_2, \quad 0 < \tau_{\mathcal{R}} \ll 1, \tag{1.2}$$

for an approximate solution $\tilde{X} \approx X$, and usually choose $\tau_{\mathcal{R}} = 10^{-8}$. We employ this residual norm based criterion because approaches based on relative changes, e.g., of the norm of the generated approximate feedback matrices $\tilde{K} = M^*\tilde{X}B$, have been proven unreliable [109]. For large-scale matrix equations, computing or estimating the residual norm is not always an easy task and can yield significant numerical effort. Recent developments reduce these costs for several of the considered methods to the level of being insignificant, though. Details on how this is achieved will be included with the descriptions of the single algorithms.

Only the first and last points arise in all discussed methods.

The memory consumption of the methods is also considered and compared. Here, it is important to distinguish between two scenarios:

S1. An approximate solution of (1.1) is sought, in the form of a low-rank approximate solution $ZDZ^* \approx X$ with solution factors $Z \in \mathbb{R}^{n\times r}$, $D \in \mathbb{R}^{r\times r}$, and $r \ll n$. This situation occurs especially in certain model order reduction techniques, e.g., LQG, positive-real, and related balanced truncation type approaches [4, 19, 44, 66, 86]. Note that in some of these model reduction approaches, the arising GCAREs can have positive definite or indefinite quadratic terms, e.g., [33]. We do not pursue this issue any further here, but these can be treated with a low-rank version of the iteration proposed in [76].

S2. Only an approximation of the stabilizing feedback matrix $K := M^*XB \in \mathbb{R}^{n\times m}$ is sought. The optimal control of linear, time-invariant, dynamical systems using LQR or LQG feedback control approaches (e.g., [59, 94]) is the prototypical application for this scenario. Since K is a very thin $n \times m$ matrix, algorithms that are able to operate solely on approximations of K are considerably more efficient regarding the memory requirements, whenever m is significantly smaller than r.

Remark 1.1. If one is only interested in the stabilizing feedback K as in scenario **S2**, alternative approaches (for $M = I$) based on the Chandrasekar differential equations

$$\dot{K}(t) = -L(t)^*(L(t)B), \qquad K(0) = 0, \tag{1.3a}$$

$$\dot{L}(t) = -L(t)(A - BK(t)^*), \qquad L(0) = C \tag{1.3b}$$

can be used [9, 53] by choosing a sufficiently large $t = t_f$ and solving the system of ordinary differential equations numerically backwards in time. The motivation behind this is that $K = \lim_{t\to-\infty} K(t)$ and $X = \lim_{t\to-\infty} \int_0^t L(s)^*L(s)\mathrm{d}s$ [37, 67]. Unfortunately, a numerical solution of (1.3) can be difficult, because the solution

trajectories often exhibit a highly oscillatory behavior for small times. An integrator may thus require a large number of tiny time steps, and it can take a long time for the above dynamical system to reach a stationary phase [92]. Therefore, we do not pursue this approach in this study but leave it as an interesting future topic, especially since it negates the need to work with an ARE at all in scenario **S2**. Similarly to the strategy in [9], one could solve (1.3) numerically until a stabilizing K is constructed, which then serves as an initial guess within an iterative method for the GCARE (1.1).

1.2. Outline. In section 2, the considered methods are briefly described and categorized into three classes. We refrain from giving detailed derivations and theoretical results for each single algorithm, as those can be found in the relevant literature. How the above computational stages arise in each method is emphasized in that section. Recent developments, which are important for the numerical performance, are also mentioned, including a few modifications so far not discussed in the literature such as, e.g., accessing the true GCARE residual norm in projection methods, defect correction strategies for occurring projected CAREs, and handling initial feedback approximations. The comparison of the computational costs is given in section 3. This addresses a single iteration step of each considered method respecting the subtasks mentioned above. The memory requirements and consumption are addressed as well, especially regarding the scenarios **S1** and **S2**. A series of numerical experiments is carried out in section 4 comparing the computation time, memory consumption, and other performance indicators of all methods. Section 5 summarizes our findings.

1.3. Notation. We use the following notation in this paper: $\mathbb{C}_-, \mathbb{C}_+$ are the open left and right half planes, $\mathrm{Re}(z)$, $\mathrm{Im}(z)$, $\overline{z} = \mathrm{Re}(z) - \jmath\,\mathrm{Im}(z)$, and $|z|$ are the real part, the imaginary part, the complex conjugate, and the absolute value of a complex quantity z, respectively. For matrices, A^* denotes the transpose for real matrices and the complex conjugate transpose for complex matrices. If it exists, A^{-1} is the inverse of A and $A^{-*} = (A^*)^{-1}$. In most situations, expressions of the form $x = A^{-1}b$ are to be understood as solving the linear system of equations $Ax = b$ for x. The relations $A \succ (\succeq) 0$ and $A \prec (\preceq) 0$ stand for the matrix A being positive and negative (semi)definite. Likewise, $A \succeq (\preceq) B$ refers to $A - B \succeq (\preceq) 0$. Unless stated otherwise, $\|\cdot\|$ is the Euclidean vector or subordinate matrix norm, and $\kappa(\cdot)$ is the associated condition number. The Frobenius norm is denoted by $\|\cdot\|_F$. For a given matrix $A \in \mathbb{R}^{n\times n}$ and a given (block) vector $v \in \mathbb{R}^{n\times p}$, the (block) Krylov subspace generated by A and v is denoted as

$$\mathcal{K}_j(A, v) := \mathrm{range}\big(\begin{bmatrix} v, & Av, & A^2v, & \ldots, & A^{j-1}v \end{bmatrix}\big).$$

2. Classification and brief introduction of the considered methods. This section briefly introduces the methods compared in the later sections. We divide the methods into three classes represented by the subsection herein. The first class consists of all methods that use a certain subspace, to project (1.1) to a much smaller dense representation, that can be solved by a direct computation. Then the solution is lifted to the full coordinates again. Direct iteration methods that successively approximate the solution, without the need for projection, form the second class. The remaining class of methods is made up of the variants of Newton's method.

2.1. Projection methods. Let $\mathcal{Q}$ be an r-dimensional subspace of $\mathbb{R}^n$ with $r \ll n$ and let the columns of $Q_r \in \mathbb{R}^{n\times r}$ form an orthonormal basis for $\mathcal{Q}$. We are looking for approximate solutions of (1.1) in the space

$$\mathcal{Z}_r(Q_r) := \{X_r = Q_r Y_r Q_r^* \ : \ Y_r^* \in \mathbb{R}^{r\times r}\}.$$

For the standard case $M = I_n$, in direct analogy to the case of Lyapunov equations [63, 93], we impose a Galerkin condition (using the Euclidean inner product) onto the CARE residual:

$$\mathcal{R}(X_r) = A^*Q_rY_rQ_r^* + Q_rY_rQ_r^*A - Q_rY_rQ_r^*BB^*Q_rY_rQ_r^* + C^*C \quad \perp \quad \mathcal{Z}_r(Q_r).$$

This condition implies that Y_r is the solution of the r-dimensional CARE

$$\tilde{A}_rY_r + Y_r\tilde{A}_r^* + Y_r\tilde{B}_r\tilde{B}_r^*Y_r + \tilde{C}_r^*\tilde{C}_r = 0 \tag{2.1}$$

with $\tilde{A}_r := Q_r^*A^*Q_r \in \mathbb{R}^{r\times r}$, $\tilde{B}_r := Q_r^*B \in \mathbb{R}^{r\times m}$, $\tilde{C}_r := CQ_r \in \mathbb{R}^{p\times r}$. For $M \neq I_n$, this Galerkin projection is typically implicitly applied to an equivalent CARE defined, e.g., by $A_M := AM^{-1}$, $B_M := B$, and $C_M = CM^{-1}$ or, if $0 \prec M = L_ML_M^*$, to $A_M := L_M^{-1}AL_M^{-*}$, $B_M := L_M^{-1}B$, and $C_M = CL_M^{-*}$. If the resulting low-rank solution X_r is not good enough, the subspace $\mathcal{Q}_r$ is expanded by additional basis vectors.

Methods following this Galerkin projection principle mainly differ in the way the subspace $\mathcal{Q}$ or, more precisely, the sequence $\mathcal{Q}_0 \subseteq \mathcal{Q}_1 \subseteq \cdots \subseteq \mathcal{Q}_j$ of subspaces, is constructed. An intuitive choice would be the block Krylov subspace generated from A_M^* and C_M^* [63]; however, because of the often resulting slow convergence, this approach has been superseded by the application of more general Krylov type subspaces.

The extended block Krylov subspace [46, 70] is given by

$$\mathcal{Q}_j = \mathcal{K}_j^{\mathrm{ext}}(A_M^*, C_M^*) := \mathcal{K}_j(A_M^*, C_M^*) \cup \mathcal{K}_j(A_M^{-1}, A_M^{-1}C_M^*).$$

In each iteration step, the subspace is expanded by $2p$ new vectors, leading to $\dim(\mathcal{Q}_j) \leq 2jp$. The basis matrix Q_j for $\mathcal{K}_j^{\mathrm{ext}}$ can be constructed by the extended block Arnoldi process which was used in [69, 100] and [61] to compute low-rank solutions of GCALEs and GCAREs, respectively. Where it does not lead to confusion, we will omit the prefix "block" in the remainder of the text and refer to algorithms using $\mathcal{K}_j^{\mathrm{ext}}$ as extended Krylov subspace methods (EKSM).

A further generalization is given by rational Krylov subspaces [90]

$$\begin{aligned}\mathcal{Q}_j &= \mathcal{K}_j^{\mathrm{rat}}(A_M^*, C_M^*, \boldsymbol{\alpha}) \\ &:= \operatorname{range}\left(\left[(A_M - \alpha_1 I)^{-*}C_M^*, \ldots, \prod_{i=1}^{j}\left((A_M - \alpha_i I)^{-*}\right)C_M^*\right]\right) \\ &= \operatorname{range}\left(\left[(A - \alpha_1 M)^{-*}M^*C_M^*, \ldots, \prod_{i=1}^{j}\left((A - \alpha_i M)^{-*}M^*\right)C_M^*\right]\right),\end{aligned}$$

where $\boldsymbol{\alpha} := \{\alpha_1, \ldots, \alpha_j\} \subset \mathbb{C}_+ \cup \{\infty\}$ are shift parameters whose selection is discussed later. Following the well-known moment matching interpretation [4], we note that $\mathcal{K}_j^{\mathrm{ext}}$ is a special case of $\mathcal{K}_{2j}^{\mathrm{rat}}$ with the shifts zero and infinity used in an alternating fashion. The usage of rational Krylov subspace methods (RKSM) for GCALEs and GCAREs was investigated in [47, 48, 101, 103]. The basic procedure of both EKSM and RKSM for (1.1) is summarized in Algorithm 1. Usually, the rational Krylov subspace is generated so that $\operatorname{range}(C_M^*) \subseteq \mathcal{Q}_j$ [47, 48, 103], which can be enforced by formally setting $\alpha_1 = \infty$.

We now give some remarks on the major steps of Algorithm 1. In lines 1, 2, 6, `orth` is to be understood as any stable (block) orthogonalization routine, such as a repeated (block) modified Gram–Schmidt process [55], which we employ in this

Algorithm 1: Extended and Rational Krylov Subspace Method for GCAREs.

Input : Matrices A, M, B, C defining (1.1) and stopping tolerance $0 < \tau_{\mathcal{R}} \ll 1$.

Output: $Q_j \in \mathbb{C}^{n\times k}$, $Y_j = Y_j^* \in \mathbb{C}^{k\times k}$ such that $Q_j Y_j Q_j^* \approx X$ with $k \ll n$ and $Q_j^* Q_j = I_k$, stabilizing feedback matrix $K_j \in \mathbb{C}^{n\times m}$.

1 $q_0 = C_M := M^{-*}C^*$, $Q_0 = \operatorname{orth}(C_M)$.

2 EKSM: $q_0^{(1)} = q_0$, solve $A^* q_0^{(2)} = M^* q_0^{(1)}$ for $q_0^{(2)}$, $Q_0 = \operatorname{orth}([Q_0, q_0^{(2)}])$.

3 **for** $j = 1, 2, \ldots, j_{\max}$ **do**

4 RKSM: Select shift $\alpha_j \in \mathbb{C}_+$.

5 Generate new basis vectors:

5a RKSM: Solve $(A - \alpha_j M)^* q_j = M^* q_{j-1}$ for q_j.

5b EKSM: Solve $M^* q_j^{(1)} = A^* q_{j-1}^{(1)}$, $A^* q_j^{(2)} = M^* q_{j-1}^{(2)}$ for $q_j^{(1)}$, $q_j^{(2)}$; set $q_j = [q_j^{(1)}, q_j^{(2)}]$.

6 Orthogonally extend basis matrix Q_{j-1}: $Q_j = \operatorname{orth}([Q_{j-1},\ q_j])$.

7 $\tilde{A}_j = Q_j^* M^{-*} A^* Q_j$, $\tilde{B}_j = Q_j^* B$, $\tilde{C}_j = C_M Q_j$.

8 Solve projected CARE $\tilde{A}_j Y_j + Y_j \tilde{A}_j^* - Y_j \tilde{B}_j \tilde{B}_j^* Y_j + \tilde{C}_j^* \tilde{C}_j = 0$ for Y_j.

9 **if** $\|\mathcal{R}(Q_j Y_j Q_j^*)\| < \tau_{\mathcal{R}} \|C^* C\|$ **then** $K_j = M^* Q_j Y_j \tilde{B}_j$, **stop**.

study. An efficient construction of the projected matrices $\tilde{A}_j$, $\tilde{C}_j$ can be found in the respective literature on EKSM [100] and RKSM [34, 48, 57, 103]. The small CARE in line 8 can be solved by direct methods involving dense numerical linear algebra; see, e.g., [13, 15, 35], whose numerical complexity is cubic in the subspace dimension.

For RKSM, the choice of shift parameters in line 4 is crucial to achieve a fast convergence. An overview of different selection strategies can be found, e.g., in [57]. The adaptive selection strategy proposed in [48] and later improved in [101] turned out to be successful in the majority of cases. There, after iteration step j of RKSM, the next shift α_{j+1} is obtained by minimizing a rational function over the convex hull of the eigenvalues of either $\tilde{A}_j$ [48], the projected closed loop matrix $\tilde{A}_j - \tilde{B}_j \tilde{K}_j^*$ with $\tilde{K}_j := Y_j \tilde{B}_j$ [81, 101], or of the matrix pair $(Q_j^* A Q_j,\ Q_j^* M Q_j)$ [51]. Here, we restrict to the variant using $\tilde{A}_j - \tilde{B}_j \tilde{K}_j^*$. It can happen that some of the generated shifts occur in complex conjugate pairs. In order to reduce the number of complex arithmetic operations, the basis matrix Q_j, and therefore also most other quantities, can be kept real by applying the real RKSM proposed in [91]. Essentially, the real RKSM consists of augmenting Q_{j-1} by $[\operatorname{Re}(q_j),\ \operatorname{Im}(q_j)]$ when $\alpha_j \in \mathbb{C} \setminus \mathbb{R}$, which is equivalent to processing both shifts α_j and $\overline{\alpha_j}$ at once. With this observation, the only remaining complex operation that occurs in the algorithm is solving the complex, sparse linear system in line 5a.

To monitor the progress and to stop the iteration, the Euclidean or Frobenius norm of the residual $\mathcal{R}_j := \mathcal{R}(Q_j Y_j Q_j^*)$ can also be computed efficiently without explicitly forming this large, dense, $n \times n$ matrix [48, 61, 100, 101, 103]. Following [48], the CARE residual in RKSM has the form (for $M = I$)

$$\mathcal{R}_j = F_j + F_j^*, \quad F_j := G_j S_j^* \in \mathbb{C}^{n\times n},$$
$$G_j := q_{j+1}\alpha_j - (I - Q_j Q_j^*) A^* q_{j+1}, \quad S_j := Q_j^* Y_j H_j^{-*} E_j h_{j+1,j}, \quad E_j = e_j \otimes I_p,$$

where $H_j = [h_{i,k}] \in \mathbb{C}^{jp\times jp}$ is block upper Hessenberg, and $h_{i,k} \in \mathbb{C}^{p\times p}$, $i = 1, \ldots, j+$

1, $k = 1, \ldots, j$, are the orthogonalization coefficients obtained in the Gram–Schmidt process. Since $Q_j^* q_{j+1} = 0$, it also holds that $S_j^* G_j = 0$ and, hence, we have the idempotence $F_j^2 = 0$. An easy calculation shows that $\|\mathcal{R}_j\| = \|F_j\| = \|\Psi h_{j+1,j} E_j^* H_j^{-1} Y_j\|$, where Ψ is the triangular factor of a thin QR factorization of G_j. The structure of G_j and S_j is slightly different in the real RKSM [91] and in EKSM [61, 100], but the overall approach remains unchanged. Note that for GCAREs ($M \neq I$) the projection method is implicitly applied to an equivalent CARE defined by matrices A_M, B_M, and C_M, such that one would obtain the norm of an associated transformed residual, e.g., $\mathcal{R}_j^M := M^{-*} \mathcal{R}_j M^{-1}$. In practice, $\|\mathcal{R}_j\|$ and $\|\mathcal{R}_j^M\|$ can differ significantly. For our comparative study of different algorithms, we prefer to monitor the residual of the original GCARE. This can be easily achieved by exploiting

$$\mathcal{R}_j = M^* \mathcal{R}_j^M M = M^* (F_j^M + (F_j^M)^*) M = N_j^M \begin{bmatrix} 0_p & I_p \\ I_p & 0_p \end{bmatrix} (N_j^M)^*, \quad F_j^M := G_j^M S_j^*,$$
$$N_j := M^* \left[G_j^M, S_j \right] \in \mathbb{C}^{n \times 2p}, \; G_j^M := q_{j+1} \alpha_j - (I - Q_j Q_j^*) A_M^* q_{j+1},$$

from which it follows that $\|\mathcal{R}_j\| = \|\Psi_M \begin{bmatrix} 0_p & I_p \\ I_p & 0_p \end{bmatrix} \Psi_M^*\|$, where Ψ_M is the triangular factor of a thin QR decomposition of N_j. In addition to the QR decomposition, $2p$ ($4p$ for complex shifts in real RKSM) matrix-vector multiplications with M^* (or factors thereof) are therefore required to compute $\|\mathcal{R}_j\|$.

2.1.1. Variants. To slow down the growth of the column dimension of Q_j in block RKSM for Lyapunov equations, a modification has been presented in [49]. There, tangential rational Krylov subspaces

$$\mathcal{Q}_j = \mathcal{K}_j^{\text{t-rat}}(A_M^*, C_M^*, \boldsymbol{\alpha})$$
$$:= \operatorname{range}\Big(\Big[(A - \alpha_1 M)^{-*} M^* C_M^* d_1, \ldots, (A - \alpha_j M)^{-*} M^* C_M^* d_j \Big] \Big), \; \alpha_i \neq \alpha_j,$$

are used to generate the projection subspace, leading to the tangential RKSM (TRKSM). It is straightforward to apply this approach to GCAREs. The tangential directions $d_j \in \mathbb{C}^{p \times p_j}$, $p_j \leq p$, are computed adaptively at every iteration step in conjunction with the shifts. Clearly, $\dim(\mathcal{K}_j^{\text{t-rat}}) = \sum_{i=1}^{j} p_i \leq jp$, leading to a reduced effort for the orthogonalization and small-scale solution of the projected matrix equation. However, as pointed out in [49], the decreased growth of $\dim(Q_j)$ can be accompanied with a slower convergence of the algorithm. Compared to the standard block RKSM, this may result in a larger number of linear systems with different coefficient matrices that need to be solved. Some of the upcoming numerical experiments confirm this.

In [64, 65], global standard and extended Krylov subspace methods for matrix equations are considered. These methods are based on a global orthogonalization process, where the constructed basis matrix $Q_j = [q_1, \ldots, q_j]$ has blocks $q_i \in \mathbb{C}^{n \times p}$ satisfying $\langle q_i, q_k \rangle = \operatorname{tr}(q_k^* q_i) = \delta_{i,k}$, $\|q_i\|_F = 1$. See also [54] for more details on these global processes. When applied for GCALEs, the projected quantities $\tilde{A}_j$, $\tilde{C}_j$ obtained in this way have special structures which can be exploited to gain savings when solving the projected small-scale problem [65, 104]. In global Krylov methods for GCAREs, however, such savings are not possible because $\tilde{B}_j$ does not reveal any special structure one could exploit. Moreover, global methods often converge slower than their block counterparts, meaning that they require a higher number of iteration steps, which results in a larger subspace dimension. In the GCARE case this leads to noticeably higher numerical costs for the solution of the projected equation. A detailed assessment and comparison of the costs of both block and global extended

Krylov subspace methods for GCALEs is given in [104], where further information on actual implementations can be found as well. For the purpose of comparison in this paper, the global EKSM (GEKSM) [64] is used in some of our numerical experiments. Global rational Krylov subspace approaches are considered in [36] in the context of model order reduction.

2.2. Nonprojective iterations. The second class of methods consists of iterative processes working directly on the Riccati equation, and circumventing the use of a Galerkin projection framework or Newton scheme as the methods in the previous and next section, respectively. The low-rank solution is built from direct relations that do not require the solution of a projected CARE. In particular, two representatives of this class are considered, both of which implicitly work with the Hamiltonian matrix

$$\mathcal{H} = \begin{bmatrix} A & -BB^* \\ -C^*C & -A^* \end{bmatrix} \in \mathbb{R}^{2n\times 2n}$$

associated to (1.1) (here we restrict ourselves to the case $M = I$ for simplicity, but the case $M \neq I$ can be developed in a straightforward manner by considering the associated Hamiltonian/skew-Hamiltonian matrix pencil). It is well known that the solution of (1.1) is given by $X = QP^{-1}$, where $[P^*, Q^*]^* \in \mathbb{R}^{2n\times n}$ spans an $\mathcal{H}$-invariant subspace with respect to the stable eigenvalues of $\mathcal{H}$. Not only is this relation the basis for many direct methods for CAREs [40, 77], but also several works [2, 16, 17, 81, 83] investigate its application in a large-scale setting.

2.2.1. The incremental low-rank subspace iteration. In [81], the Cayley iteration

$$\begin{bmatrix} P_j \\ Q_j \end{bmatrix} = (\mathcal{H} + \alpha_j I_{2n})^{-1}(\mathcal{H} - \overline{\alpha}_j I_{2n}) \begin{bmatrix} I_n \\ X_{j-1} \end{bmatrix}, \quad X_j^{\text{cay}} = Q_j P_j^{-1}, \tag{2.2}$$

with shift parameters $\alpha_j \in \mathbb{C}_+$, is proposed for the Hamiltonian matrix $\mathcal{H}$ to iteratively compute the desired $\mathcal{H}$-invariant subspace. For feasibility in the large-scale case, one can observe that if the initial approximation $X_1^{\text{cay}} \succeq 0$ has rank p, then X_j^{cay} has rank jp, and it is possible to rewrite the above iteration into a low-rank version [81], incrementally generating a low-rank approximation $X \approx X_j^{\text{cay}} = Z_j Y_j^{-1} Z_j^*$ with $Z_j \in \mathbb{C}^{n\times jp}$, $0 \prec Y_j = Y_j^* \in \mathbb{C}^{jp\times jp}$. This method is referred to as the Incremental Low-Rank Subspace Iteration (ILRSI) and is illustrated in Algorithm 2.

Interestingly, the same iteration has been derived independently in [83], with a completely different approach, by analyzing an optimal control problem associated to (1.1). Only the setup of the middle factor Y_j is done slightly differently in [83]. If $B = 0$, ILRSI reduces to the low-rank alternating direction implicit (LR-ADI) iteration [27, 79, 89] for GCALEs. The shift parameters α_j are, once again, crucial for the speed of the convergence and may be selected in advance or, preferably, adaptively by strategies borrowed from the GCALE case, e.g., [24, 108]. Approaches that take the GCARE structure into account more will be discussed later. The relation to the LR-ADI iteration can be exploited to deal with pairs of complex conjugate shifts since [23, Theorem 1] works for Algorithm 2 as well: For $\alpha_j \in \mathbb{C}_+$, $\alpha_{j+1} = \overline{\alpha_j}$, it holds that $V_{j+1} = \overline{V_j} - \beta_j \operatorname{Im}(V_j)$ with $\beta_j := 2\frac{\operatorname{Re}(\alpha_j)}{\operatorname{Im}(\alpha_j)}$ and $Z_{j+1} = [Z_{j-1}, \operatorname{Re}(V_j) - \beta_j \operatorname{Im}(V_j), \sqrt{(\beta^2+1)}\operatorname{Im}(V_j)]$. Hence, for each complex pair, a double iteration step can be performed by solving only one complex linear system.

The most costly numerical work in ILRSI appears to be the solution of the shifted linear systems with p right-hand sides in line 2. However, the estimation of the residual

Algorithm 2: Incremental Low-Rank Subspace Iteration (ILRSI).

Input : Matrices $A,\ M,\ B,\ C$ defining (1.1)
Output: $Z_j \in \mathbb{C}^{n\times jp}$, $Y_j \in \mathbb{C}^{jp\times jp}$ s.t. $Z_j Y_j^{-1} Z_j^* \approx X$.

1 Determine shifts $\{\alpha_1, \ldots, \alpha_{j_{\max}}\} \subset \mathbb{C}_+$.
2 Solve $(\alpha_j M^* - A^*)V_1 = C^*$ for V_1.
3 Update solution factors $Y_1 = 2\operatorname{Re}(\alpha_1)(I + V_1^* B B^* V_1)$, $Z_1 = V_1$.
4 **for** $j = 2, 3, \ldots, j_{\max}$ **do**
5 Solve $(\alpha_j M^* - A^*)\tilde{V} = M^* V_{j-1}$ for $\tilde{V}$, $V_j = V_{j-1} - (\alpha_j + \overline{\alpha_{j-1}})\tilde{V}$.
6 Update low-rank solution factor $Z_j = [Z_{j-1}\ \ V_j]$.
7 $L_j = \begin{bmatrix} I_{jp-1} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \\ & \ddots & \vdots \\ & & 1 \end{bmatrix} \begin{bmatrix} \frac{\overline{\alpha_1}+\alpha_j}{2\operatorname{Re}(\alpha_j)} & & & \\ \frac{\alpha_1-\alpha_j}{2\operatorname{Re}(\alpha_j)} & \frac{\overline{\alpha_2}+\alpha_j}{2\operatorname{Re}(\alpha_j)} & & \\ & \ddots & \ddots & \\ & & \frac{\alpha_{j-1}-\alpha_j}{2\operatorname{Re}(\alpha_j)} & \frac{\overline{\alpha_j}+\alpha_j}{2\operatorname{Re}(\alpha_j)} \end{bmatrix} \otimes I_p$.
8 $N_j = L_j^{-1} + \operatorname{diag}(I, 0)$, $V_B = L_j^{-*} V_j^* B$.
9 Update middle low-rank factor $Y_j = N_j^{-*}\left(\begin{bmatrix} Y_{j-1} & \\ & 2\operatorname{Re}(\alpha_j) \end{bmatrix} + \frac{1}{2\operatorname{Re}(\alpha_j)} V_B V_B^*\right) N_j^{-1}$.
10 **if** $\|\mathcal{R}(Z_j Y_j^{-1} Z_j^*)\| < \tau_{\mathcal{R}} \|C^* C\|$ **then stop**.
11 $j = j + 1$

norm $\|\mathcal{R}(Z_j Y_j^{-1} Z_j^*)\|$ in line 10 is potentially expensive as well. The reason is that, unlike the LR-ADI iteration for GCALEs [23] or the other discussed methods for GCAREs, there is no factorization of the GCARE residual with fixed rank known for ILRSI, e.g., of the form $\mathcal{R}(Z_j Y_j^{-1} Z_j^*) = W D W^*$ with fixed sized W, D and $\operatorname{coldim}(W) \ll n$. Of course, one can always use $\mathcal{R}(Z_j Y_j^{-1} Z_j^*) = \hat{W} \hat{D} \hat{W}^*$ with residual factors

$$\hat{W} := [A^* Z_j, M^* Z_j, C^*] \in \mathbb{C}^{n\times(2j+1)p},$$
$$\hat{D} := \operatorname{diag}\left(\begin{bmatrix} 0 & Y_j^{-1} \\ Y_j^{-1} & Y_j^{-1} Z_j^* B B^* Z_j Y_j^{-1} \end{bmatrix}, I_p\right) \in \mathbb{C}^{(2j+1)p\times(2j+1)p},$$

whose (column) dimensions obviously increase as the iteration proceeds. Computing the residual norm via a factorization like that (see, e.g., [27]) would be quickly getting extraordinarily expensive both memorywise and computationwise. Currently, one of the most reasonable things one can do is to exploit that the spectral norm of a symmetric matrix coincides with its largest eigenvalue and apply a Lanczos process implicitly to $\mathcal{R}(Z_j Y_j^{-1} Z_j^*)$. Because of the way the middle low-rank factor Y_j is constructed, computing an approximate feedback matrix K_j requires the complete already computed low-rank solution. Therefore, like the projection methods of the previous section, ILRSI does not gain an advantage with respect to memory consumption in the scenario **S2**.

2.2.2. The RADI iteration. Assume that the equation (1.1) has an approximate solution $X_1 = \Xi_1 \succeq 0$ with a positive semidefinite residual $\mathcal{R}(X_1) = R_1 R_1^*$. Then the exact solution X can be written as $X = X_1 + \tilde{X}_1$, where $\tilde{X}_1$ is a unique stabilizing solution [17, Theorem 1] of the residual Riccati equation

$$A_1^* \tilde{X}_1 + \tilde{X}_1 A_1 + C_1^* C_1 - \tilde{X}_1 B B^* \tilde{X}_1 = 0, \tag{2.3}$$

with $A_1 = A - BK_1^*$, $C_1 = R_1^*$, $K_1 = X_1 B$ (once more we suppose $M = I$ for ease of presentation). This is again a Riccati equation, so one can repeat the procedure and find an approximate solution $\Xi_2 \succeq 0$ for (2.3) such that its residual with respect to this equation is positive semidefinite, and accumulate the approximation $X_2 = X_1 + \Xi_2$, etc. This way, we obtain an increasing sequence of approximations $0 \preceq X_1 \preceq X_2 \preceq \cdots \preceq X$ to the solution of the original equation, all of which will have positive semidefinite residuals.

It remains to explain how to construct an approximate solution with a positive semidefinite residual in the jth step of the above sequence. One possibility is to carry out one step of the Cayley iteration (2.2) for the residual equation with the Hamiltonian matrix

$$\mathcal{H}_j = \begin{bmatrix} A_j & -BB^* \\ -C_j^* C_j & -A_j^* \end{bmatrix} = \begin{bmatrix} A - BK_j^* & -BB^* \\ -R_j R_j^* & -(A - BK_j^*)^* \end{bmatrix};$$

here $K_j = X_j B$. Setting the initial approximation in (2.2) to zero, after some calculation we obtain that the first step approximation Ξ_{j+1} in the Cayley–Hamilton iteration is given by

$$\begin{aligned} \Xi_{j+1} &= -2\,\mathrm{Re}(\alpha_j) V_{j+1} \tilde{Y}_{j+1}^{-1} V_{j+1}^*, \\ V_{j+1} &= (A - BK_j^* + \alpha_j I)^{-*} R_j, \\ \tilde{Y}_{j+1} &= I + (V_{j+1}^* B)(V_{j+1}^* B)^*, \end{aligned}$$

and that the residual factor and the feedback matrix can be updated as

$$\begin{aligned} R_{j+1} &= R_j - 2\,\mathrm{Re}(\alpha_j) V_{j+1} \tilde{Y}_{j+1}^{-1}, \\ K_{j+1} &= K_j - 2\,\mathrm{Re}(\alpha_j)(V_{j+1} \tilde{Y}_{j+1}^{-1})(V_{j+1}^* B). \end{aligned}$$

This describes one step of the RADI procedure. The whole method for generalized CAREs is displayed in Algorithm 3. Note that, unlike the projection methods, each RADI iteration takes the same amount of time and does not get progressively slower as the algorithm proceeds: the matrices V_j, K_j, R_j remain of the same size throughout the iteration.

The shift parameters α_j are again important for a rapid convergence. We mention and later on employ an approach proposed in [16, 17]: Let $M = I$, $U_\ell \in \mathbb{R}^{n \times \ell p}$, $U_\ell^* U_\ell = I_\ell$, $\ell p \ll jp$ with $\mathrm{range}(U_\ell) \subset \mathrm{range}(Z_j)$ and let

$$\mathcal{H}_j^{\mathrm{proj}} = \begin{bmatrix} U_\ell^* A_j U_\ell & -(U_\ell^* B)(U_\ell^* B)^* \\ -(U_\ell^* R_j^*)(U_\ell^* R_j^*)^* & -U_\ell^* A_j^* U_\ell \end{bmatrix} \in \mathbb{R}^{2\ell \times 2\ell}$$

be the Hamiltonian matrix associated to the residual CARE projected onto $\mathrm{range}(U_\ell)$. If $\theta_i, q_i = \begin{bmatrix} q_i^{(1)} \\ q_i^{(2)} \end{bmatrix}$, with $q^{(1,2)} \in \mathbb{C}^\ell$, are the eigenpairs of $\mathcal{H}_j^{\mathrm{proj}}$, then the next shift α_{j+1} is selected as the eigenvalue θ_i with the largest value of $\|q_i^{(2)}\|$. The motivation behind this strategy, along with further details, can be found in [16, 17]. In the general case $M \neq I$, the eigenpairs of the corresponding (projected) Hamiltonian/skew-Hamiltonian pencil have to be considered. The number of block columns ℓ taken into account is typically chosen very small; e.g., one takes the last $2p$ columns of Z_j. This shift selection strategy appears to be efficient and, judging by the numerical experiments in [17], often superior to other approaches. Pairs of complex conjugated shifts

Algorithm 3: The RADI Iteration for GCAREs.

Input : Matrices $A,\ M,\ B,\ C$ defining (1.1), initial feedback K_0
Output: $Z_j \in \mathbb{C}^{n\times jp}$, $Y_j = Y_j^* \in \mathbb{C}^{jp\times jp}$ s.t. $Z_j Y_j^{-1} Z_j^* \approx X$, stabilizing feedback matrix $K_j \in \mathbb{C}^{n\times m}$.

$R_0 = C^*$, $Y_0 = [\,] = [\,]$, $j = 1$.
for $j = 1, 2, \ldots, j_{\max}$ **do**
  Determine shift $\alpha_j \in \mathbb{C}_-$, $\gamma_j := \sqrt{-2\,\mathrm{Re}(\alpha_j)}$.
  Solve $(A - BK_{j-1}^* + \alpha_j M)^* V_j = R_{j-1}$ for $V_j \in \mathbb{C}^{n\times p}$.
  $V_B = V_j^* B$, $V_+ = \gamma_j^2 M^* V_j$, $\tilde{Y}_j = I + V_B V_B^*$.
  If required, update low-rank solution factors $Z_j = [Z_{j-1}\ \ \gamma_j V_j]$, $Y_j = \begin{bmatrix} Y_{j-1} & \\ & \tilde{Y}_j \end{bmatrix}$.
  Update Riccati residual factor $R_j = R_{j-1} + (V_+ \tilde{Y}_j^{-1}) \in \mathbb{C}^{n\times p}$.
  Update feedback matrix $K_j = K_{j-1} + (V_+ \tilde{Y}_j^{-1}) V_B \in \mathbb{C}^{n\times m}$.
  if $\|R_j^* R_j\| < \tau_{\mathcal{R}} \|C^* C\|$ **then stop**.

can be dealt with appropriately [17] so that only an absolutely necessary number of complex arithmetic operations remains and, e.g., the Z_j, Y_j, K_j will be real matrices.

In contrast to ILRSI, because of the block-diagonal structure of the constructed middle low-rank factor Y_j, it is possible to accumulate only the feedback approximation K_j in the RADI iteration without ever forming the low-rank factors Z_j, Y_j. This makes the method a good memory-efficient candidate for scenario **S2**.

2.2.3. Equivalences and relations to other methods. As shown in [17, Theorem 2], if executed with the same set of shift parameters, ILRSI and the RADI iteration both produce the same output and are, in fact, equivalent. Hence, both methods are simply different implementations of the same approximation sequence. Moreover, if $B = 0$, both methods reduce to the LR-ADI iteration for Lyapunov equations. Because of this close relation, we will also execute ILRSI with the residual Hamiltonian shifts described above. ILRSI and the RADI iteration are in the same sense also equivalent to the quadratic ADI iteration [111]. We will not incorporate the quadratic ADI iteration into this study since it has been shown to be inferior in terms of numerical efficiency to the considered methods; see, e.g., [16, 17].

Although no Galerkin projection is used within ILRSI, RADI, it can still be shown that the computed low-rank solution factors Z_j span a rational Krylov subspace. In fact, since in ILRSI the construction of the columns in Z_j is very close to the LR-ADI iteration for Lyapunov equations, it can be shown that $\mathrm{range}(Z_j) = \mathrm{range}\Big([(A-\alpha_1 M)^{-*} C^*, \ldots, [\prod_{i=2}^{j}(A-\alpha_i M)^{-1}](A-\alpha_1 M)^{-1} C^*]\Big)$; see, e.g., [78, 79, 110]. Further connections of ILRSI to the rational Krylov framework are discussed in [83]. By the above equivalence, those connections hold for RADI as well.

2.3. Low-rank Newton methods. Motivated by the nonlinear nature of $\mathcal{R}(X)$, Newton schemes are applied to solve (1.1). With a given initial guess X_0, step k of a Newton iteration for (1.1) is, following, e.g., [6, 74], given by

$$\mathcal{R}'|_{X_{k-1}}(N_k) = -\mathcal{R}(X_{k-1}), \quad X_k = X_{k-1} + N_k. \tag{2.4}$$

Here, $\mathcal{R}'|_X$ denotes the Fréchet derivative of $\mathcal{R}$ at X and is given by

$$\mathcal{R}'|_X \ : \ N \mapsto (A - BB^*XM)^*NM + M^*N(A - BB^*XM).$$

Hence, the update N_k is the solution of a generalized, algebraic, continuous-time Lyapunov equation (GCALE). Kleinman [68] showed that it is possible to reformulate the Newton step to directly compute X_k instead of working with the correction N_k. In that case, the iteration and the corresponding GCALE to be solved are

$$A_k^*X_kM + M^*X_kA_k = -F_kF_k^*, \quad F_k := [C^*, K_{k-1}], \quad A_k := A - BK_{k-1}^* \tag{2.5a}$$

$$\text{with} \quad K_{k-1} := M^*X_{k-1}B. \tag{2.5b}$$

Under the assumed properties of this paper and provided X_0 is a stabilizing initial guess, the sequence $\{X_k\}_{k\geq 1}$ converges quadratically towards X, the exact solution of (1.1). Moreover, it converges monotonically decreasing with respect to the Löwner ordering, i.e., $X_1 \succeq X_2 \succeq \cdots \succeq X$, and for all $k > 0$ it holds that $\Lambda(A - BB^*X_kM, M) \subset \mathbb{C}_-$. Because of the size restriction on B, C we set in the introduction, the right-hand side F_k in (2.5a) is (at most) of rank $p+m \ll n$. We can, therefore, utilize low-rank algorithms for large-scale GCALEs [32, 102] that compute low-rank solution factors of the solution X_k in (2.5a), e.g., $X_k \approx Z_kZ_k^*$. Since these algorithms are typically of an iterative nature, we end up with an inner-outer iteration, consisting of the inner particular GCALE solver iteration, and the outer Newton iteration. To distinguish these two stages, subscripts $_k$ and bracketed superscripts $^{(j)}$ will refer to data associated to the outer ($_k$) and inner ($^{(j)}$) iterations. In many papers [20, 27, 30, 31, 32, 52], the low-rank alternating direction implicit (LR-ADI) iteration [22, 79, 89] is employed for the purpose of solving the GCALE (2.5a). For a fixed $k \geq 1$, the LR-ADI iteration in the most recent formulation [22, 72] proceeds for $j \geq 1$ in the following form:

$$\begin{aligned} V_k^{(j)} &= (A_k + \alpha_k^{(j)}M)^{-*}W_k^{(j-1)}, \quad W_k^{(j)} = W_k^{(j-1)} - 2\,\mathrm{Re}(\alpha_k^{(j)})M^*V_k^{(j)}, \\ Z_k^{(j)} &= \left[Z_k^{(j-1)},\ \sqrt{-2\,\mathrm{Re}(\alpha_k^{(j)})}V_k^{(j)}\right], \end{aligned} \tag{2.6}$$

where $W_k^{(0)} := F_k$, $Z_k^{(0)} = [\,]$, and $\alpha_k^{(j)} \in \mathbb{C}_-$ are shift parameters. Using (2.6) as the inner Lyapunov solver within the Newton–Kleinman iteration (2.5) leads to the low-rank Newton–Kleinman ADI (NK-ADI) method, which is illustrated in Algorithm 4.

The shift parameters $\alpha_k^{(j)}$ in line 5 are, similarly to those in RKSM, ILRSI, and RADI, crucial for a fast reduction of the error. Here, we employ, without going into detail, the automatic shift generation strategy proposed in [24]: first, several shifts are selected from the spectrum of the matrix pair $(U_\ell^*A_kU_\ell,\ U_\ell^*MU_\ell)$, where $U_\ell \in \mathbb{R}^{n\times \ell p}$, $U_\ell^*U_\ell = I_\ell$, $\ell < j$, with $\mathrm{range}(U_\ell) \subset \mathrm{range}\left(Z_k^{(j)}\right)$. Once these shifts are depleted, the procedure is repeated. This shift selection strategy usually leads to the best performance of the LR-ADI iteration, both in terms of execution time and the required number of iteration steps. Pairs of complex shifts can be handled as in the LR-ADI [22, 23]. It can be shown [20, 32, 72] that, inherited from the LR-ADI iteration [22], the GCARE residual at inner step j and outer step k is given by

$$\mathcal{R}_k^{(j)} := \mathcal{R}(Z_k^{(j)}(Z_k^{(j)})^*) = [W_k^{(j)},\ \Delta K_k^{(j)}]\,\mathrm{diag}(I_{p+m}, -I_m)\,[W_k^{(j)},\ \Delta K_k^{(j)}]^*,$$

Algorithm 4: Low-rank Newton–Kleinman ADI for GCAREs (NK-ADI).

Input : Matrices $A,\ M,\ B,\ C$ defining (1.1), initial feedback K_0, and stopping tolerances $0 < \tau_{\mathrm{ADI}}, \tau_{\mathcal{R}} \ll 1$.
Output: $Z_k \in \mathbb{C}^{n\times(m+p)j}$ such that $Z_k Z_k^* \approx X$, stabilizing feedback matrix $K_k \in \mathbb{C}^{n\times m}$.

1 **for** $k = 1, \ldots, k_{\max}$ **do**
2 $\quad W_k^{(0)} = [C^*, K_{k-1}]$, $Z_k^{(0)} = [\,]$, $K_k^{(0)} = 0$, $j = 0$.
3 $\quad$ **while** $\|W_k^{(j)}\|^2 > \tau_{ADI}\|W_k^{(0)}\|^2$ **do**
4 $\quad\quad j = j + 1$
5 $\quad\quad$ Determine next shift $\alpha_k^{(j)}$.
6 $\quad\quad$ Solve $(A_k + \alpha_k^{(j)} M)^* V_k^{(j)} = W_k^{(j-1)}$ for $V_k^{(j)}$.
7 $\quad\quad \gamma_k^{(j)} = \sqrt{-2\,\mathrm{Re}(\alpha_k^{(j)})}$, $V_+ := (\gamma_k^{(j)})^2 M^* V_k^{(j)}$.
8 $\quad\quad$ Update factor of Lyapunov residual $W_k^{(j)} = W_k^{(j-1)} + V_+$.
9 $\quad\quad$ Update low-rank solution factor $Z_k^{(j)} = [Z_k^{(j-1)},\ \gamma_k^{(j)} V_k^{(j)}]$ if required.
10 $\quad\quad$ Implicit update of feedback matrix $K_k^{(j)} = K_k^{(j-1)} + V_+ (V_k^{(j)})^* B$.
11 $\quad K_k = K_k^{(j)}$, $Z_k = Z_k^{(j)}$.
12 $\quad$ **if** $\|\mathcal{R}(Z_k Z_k^*)\| < \tau_{\mathcal{R}}\|C^*C\|$ **then stop**.

where $\Delta K_k^{(j)} := K_k^{(j)} - K_{k-1} \in \mathbb{C}^{n\times m}$ and $W_k^{(j)} \in \mathbb{C}^{n\times(p+m)}$ is the explicit low-rank factor of the current GCALE residual. Hence, computing the GCARE residual norm requires only the computation of the spectral norm of a thin $n \times (p + 2m)$ matrix.

Algorithm 4 reveals another advantage of using the LR-ADI iteration to solve the GCALE (2.5a). Similar to the RADI algorithm, it is not necessary to store the low-rank factor $Z_k^{(j)}$ (line 9) if only the feedback matrix K is of interest, since $K_k^{(j)}$ can be updated incrementally (cf. line 10, [27]). Thus, the NK-ADI method is also very memory efficient in scenario **S2**. Note that, in order to execute the proposed shift strategy, a small number of additional columns needs to be stored to carry out the projection of $(A_k,\ M)$.

Naturally, one could also use a projection method like EKSM or RKSM [48, 100] for solving the Lyapunov equation (2.5a), which leads to other interesting results [103], e.g., regarding the Riccati residual. However, doing so would sacrifice the advantage in scenario **S2** since, just as with the projection methods in section 2.1, the matrices $Z_k^{(j)}$ would need to be stored.

Regardless of the low-rank method employed to solve (2.5a), the low-rank Newton method in the presented form often shows a comparatively slow convergence towards an approximate stabilizing solution. One intuitive explanation is that the method updates the feedback gain approximations only after an outer iteration is completed. In practice, one of the following improvements is therefore usually mandatory to make the low-rank Newton method competitive.

2.3.1. Galerkin acceleration of the outer iteration. Similar to RADI, ILRSI, it holds that $\mathrm{range}\left(Z_k^{(j)}\right) = \mathcal{K}_j^{\mathrm{rat}}(A_k, M, [C^*, K_{k-1}], \boldsymbol{\alpha}_k)$ (see [78, 79, 110]), so in order to improve the convergence of the Newton iteration, [31, 94] suggest performing a Galerkin projection onto $\mathrm{range}(Z_k)$ in each outer iteration step, just after

the GCALE (2.5a) has been solved. Assume the columns of $Q_k \in \mathbb{C}^{n\times j(p+m)}$ with $Q_k^*Q_k = I$ constitute an orthonormal basis for range(Z_k). The projection is performed similarly to the methods in section 2.1: a small-scale, at most $j(p+m)$-dimensional GCARE

$$\tilde{A}_k^* Y_k \tilde{M}_k + \tilde{M}_k^* Y_k \tilde{A}_k^* - \tilde{M}_k^* Y_k \tilde{B}_k \tilde{B}_k^* Y_k \tilde{M}_k + \tilde{C}_k^* \tilde{C}_k = 0$$

with $\tilde{A}_k := Q_k^* A Q_k$, $\tilde{B}_k = Q_k^* B$, $\tilde{C}_k = C Q_k$, $\tilde{M}_k := Q_k^* M Q_k$ is solved for its stabilizing solution Y_k. The restriction $\tilde{A}_k$ can be constructed without matrix vector multiplications involving A. The precise formula depend on the choice of the low-rank Lyapunov solver, e.g., the LR-ADI iteration [72] or projection methods [48, 103]. The approximate solution of (1.1) associated to the Galerkin acceleration is $X_k^{\mathrm{pr}} = Q_k Y_k Q_k^*$. The next Newton–Kleinman iteration step then continues with the updated feedback matrix $K_{k+1}^{\mathrm{pr}} := M^* X_k^{\mathrm{pr}} B = M^* Q_k Y_k \tilde{B}_k$. The GCARE residual $\mathcal{R}(X^{\mathrm{pr}})$ after this projection does not have the low-rank structure from above, and, thus, $\|\mathcal{R}(X_k^{\mathrm{pr}})\|$ has to be computed differently, e.g., via applying a Lanczos process to $\mathcal{R}(X_k^{\mathrm{pr}})$ similar to ILRSI. Moreover, in order to perform this Galerkin projection, the whole low-rank solution factor Z_k has to be stored, so this variant of the NK-ADI method loses its ability to operate solely on feedback approximations and the advantage for scenario **S2**. However, numerical experiments [26, 31, 72, 94] show that this projection tremendously accelerates the outer Newton–Kleinman iteration. Often, it decreases the number of required Newton steps down to one or two. We will abbreviate this approach by NK-ADI+GP.

Remark 2.1. In [31, 94] it is also suggested to accelerate the inner low-rank ADI iteration of Algorithm 4 by a Galerkin projection framework. In view of the improvements in the LR-ADI formulation, we strongly refrain from this idea. On the one hand, using a Galerkin projection within the LR-ADI iteration destroys important structural properties of the method, e.g., the associated GCALE residual no longer has low-rank factorization, and it is then not clear what should be used as $W_k^{(j)}$ in (2.6). On the other hand, one could have used a projection method as an inner iteration from the start as discussed in [103].

2.3.2. Inexact GCALE solves and line-search. Newton methods are, under some mild conditions, still able to converge to the desired solution if the Newton step is carried out inexactly; see, e.g., [43, 50]. For the Newton–Kleinman iteration (2.5) for GCAREs, this means that the GCALE (2.5a) is only solved inexactly; for example, the LR-ADI iteration might be terminated once it satisfies $\|A_k^* X_k M + M^* X_k A_k + F_k F_k^*\| \le \tau_{\mathrm{ADI}}$ for an appropriately chosen $\tau_{\mathrm{ADI}} > 0$; e.g., $\tau_{\mathrm{ADI}} = \eta\|\mathcal{R}_{k-1}\|$, $\eta \in (0,1)$. These inexact solves, obviously, have the potential to drastically reduce the number of required LR-ADI steps. At the same time, convergence towards a stabilizing solution of the GCARE (1.1) has to be maintained, which makes the analysis difficult [20, 52, 62, 109]. In [20, 109], a novel theoretical and numerical framework for the inexact Newton–Kleinman iteration for (1.1) is proposed which additionally incorporates a line search strategy [18]. At Newton step k, let K_k be a feedback approximation linked to the approximate or exact solution $X_k^{(j)} = Z_k Z_k^*$ of the GCALE (2.5a). Then the next Newton step is carried out with an improved feedback approximation $\hat{K}_k = (1-\beta)K_{k-1} + \beta K_k$ for an appropriately chosen $\beta \in (0,1]$. The numerical effort to carry out this line search strategy is negligible, and we refer the reader to [20, 109] for details about its practical implementation, including possible choices of η, β. Compared to the plain NK-ADI iteration (Algorithm 4), the number of both inner

and outer iteration steps, and consequently the number of arising linear systems, can be drastically reduced by this approach. The inexact NK-ADI method equipped with this line-search strategy (abbreviated by iNK-ADI+LS) is still able to work only on the feedback approximations K_k, preserving the advantage of NK-ADI in scenario **S2**.

2.4. Related and further methods. Apart from the methods described so far, a number of other methods for solving large-scale GCAREs can also be found in the literature.

The structured doubling algorithm (SDA) [35, 41] is a recent method based on efficient computation of deflating subspaces by doubling. The original formulation of the method is suited for small-scale dense CAREs, for which it performs very well. The SDA has also been adapted to the large-scale setting [41, 80]. However, our experience with a number of numerical experiments indicates that the large-scale algorithm is not yet competitive with the methods described in this paper. Just as an illustration, we included the results obtained by the large-scale SDA in Example 4.4 of section 4. We have therefore chosen to omit this method in the detailed analysis.

Another class of methods is those that compute eigenvectors of the Hamiltonian matrix $\mathcal{H}$ associated with the stable eigenvalues [2, 16]. As mentioned in section 2.2, the exact solution of the CARE is $X = QP^{-1}$, where $[P^*, Q^*]^* \in \mathbb{R}^{2n\times n}$ spans the $\mathcal{H}$-invariant stable subspace. Computing the entire stable subspace of a large matrix is not feasible, so these methods target a small number $\ell \ll n$ of eigenvectors and use only these to approximate X by means of carefully designed formulas [2]. The main issue of this approach is that it is quite difficult to determine which eigenvectors should be targeted [16]. This difficulty renders the eigenvector-based methods noncompetitive as well. On the other hand, these methods have motivated the development of both ILRSI and RADI. The original derivation of RADI [17] uses the formula from [2] to find an approximate solution Ξ_1 to (2.3); the goal of ILRSI is computation of the stable subspace of $\mathcal{H}$ as well, although it is achieved by different means. For $p = 1$, ILRSI/RADI can be considered a generalization of the methods from [2, 16], since they are equivalent to these methods when eigenvalues of $\mathcal{H}$ are used as shifts [17].

2.5. Unstable GCAREs. In the literature regarding large-scale GCAREs, often only the stable case is used to test the numerical methods. We briefly discuss the situation where the GCARE (1.1) is defined by an unstable matrix pencil $A - \lambda M$. Recall that the overall system (A, M, B) has to be stabilizable. One way of handling this situation is to provide a stabilizing initial guess $X_0 = X_0^* \in \mathbb{R}^{n\times n}$, meaning that the pair $(A_{K_0} := A - BB^*X_0^*M, M)$ is stable. Also, depending on how serious the instability of $A - \lambda M$ is, some of the investigated methods might be able to converge without an initial guess; see [101] for results regarding projection methods. Of course, providing an initial guess can also help to speed up the whole method, independent of the stability of the pencil $A - \lambda M$; see [103] for some numerical tests on this.

Using an additive decomposition $X = X_+ + X_0$ of the sought-after stabilizing solution of (1.1), it is well known [15, 84] that the increment $X_+ = X_+^* \in \mathbb{R}^{n\times n}$ is the stabilizing solution of the stable GCARE

$$A_{K_0}^* X_+ M + M^* X_+ A_{K_0} - M^* X_+ BB^* X_+ M + \mathcal{R}(X_0) = 0. \tag{2.7}$$

The stabilized GCARE (2.7) can now be dealt with by using any of the Riccati methods described so far, providing A_{K_0}, M, B, and $\mathcal{R}(X_0)$ as the new input. In the case of RKSM, EKSM, and ILRSI, the altered structure of the input matrices leads

to an increase in the algorithm complexities, especially regarding the linear systems solves. In contrast to that, the RADI iteration and the low-rank Newton methods can be employed to the original GCARE (1.1) right away if the initial feedback K_0 is given to them as well.

Strategies for computing an initial feedback K_0 with stabilizing property can be found, e.g., in [3, 9, 11, 20, 35, 59, 74, 92, 99, 109]. In the remainder, we only use the approach from [3, 11], which we briefly describe next.

For $M = I$, let $A^*Q_u = Q_uR_u$ be a partial real Schur decomposition of A^* such that $\Lambda(R_u) = \Lambda(A) \cap \mathbb{C}_+$, i.e., the columns of Q_u form an orthonormal basis of the associated unstable invariant subspace. An initial guess is then defined with $X_0 := Q_uS_uQ_u^*$ and $K_0 := X_0B$, where S_u solves the algebraic Bernoulli equation $R_uS_u + S_uR_u^* - S_u(Q_u^*B)(B^*Q_u)S_u = 0$. Assuming that the number of unstable eigenvalues is small ($u \ll n$), this Bernoulli equation can be solved by standard, dense methods [12]. As an alternative, one can, under some mild conditions [3], use $X_0 := Q_uT_u^{-1}Q_u^*$, $K_0 = Q_uT_u^{-1}Q_u^*B$ with T_u being the solution of the small Lyapunov equation $-R_uT_u - T_uR_u^* + (Q_u^*B)(B^*Q_u) = 0$. This equation is potentially easier to solve than the Bernoulli equation. As an important side effect, the initial Riccati residual in (2.7) satisfies $\mathcal{R}(X_0) = C^*C$ with both of these choices. It is noteworthy that with this selection of an initial guess, it is in scenario **S2** sufficient to provide just the feedback matrix $K_0 := M^*X_0B \in \mathbb{R}^{n\times m}$ instead of X_0.

Remark 2.2. Originally, the left and right eigenvectors of A corresponding to $\Lambda(A) \cap \mathbb{C}_+$ were used in the computation of X_0 [3, 59]; this still leads to the same initial guess as above. We prefer the usage of invariant subspaces since computing a partial Schur decomposition of a large matrix A is often easier than computing left and right eigenvectors.

Nevertheless, computing the entire unstable invariant subspace and, especially, ensuring that no unstable eigenvalues are missed can in practice be a very demanding task. Large-scale eigenvalue methods for this purpose can be found, e.g., in [7] and the references therein.

In the numerical experiments, whenever unstable CAREs are considered, we assume for the sake of simplicity that the required initial feedback is provided and do not consider the numerical effort and difficulties of its computation. The latter, however, is nevertheless a crucial subject to be addressed in future research, where, e.g., strategies from general Newton and homotopy methods for an initial guess selection might be adopted [1].

3. Comparison of the main computational stages. In this section, we analyze the major computational subtasks that arise in the discussed algorithms: solving large linear systems of equations with multiple right-hand sides, computing shift parameters, building orthogonal bases, and solving projected CAREs, as well as computing, or estimating, the CARE residual norm.

3.1. Solving linear systems. The most prominent feature of all considered algorithms is the solution of a large linear system of equations with possibly multiple right-hand sides. We here restricted ourselves to sparse direct solution strategies because, on the one hand, this worked sufficiently well in our experiments. On the other hand, due to the sheer number of different iterative solvers and preconditioning strategies, a thorough discussion of preconditioned iterative solves would clutter the presentation and is, therefore, beyond the scope of this study. Also, the occurrence of multiple right-hand sides further increases the number of available iterative solution

approaches. Moreover, the effect of errors arising in the solution of linear systems on the methods for large matrix equations needs to be considered in the case of iterative linear solvers. Some research on this topic for large Lyapunov equations can be found in [73, 97, 105].

An overview of the structure of the arising systems in stable and unstable situations, as well as the number of columns in the right-hand sides, is given in Table 1. In several GCARE methods, we notice the occurrence of systems defined by the sum of a sparse matrix $A+\alpha_j M$ and a low-rank term BK_j^* of rank at most m. If sparse-direct solvers are to be applied, such systems can be dealt with by the Sherman–Morrison–Woodbury (SMW) formula [55] via

$$y_j = (A - BK_j^* + \alpha_j M)^{-*} b_j = w_j + g_j(I_m - B^* g_j)^{-1} B^* w_j,$$
$$\text{where } [w_j, g_j] \text{ solves } (A+\alpha_j M)^*[w_j, g_j] = [b_j, K_j].$$

Hence, solving a sparse-plus-low-rank system with s right-hand sides is expressed as solving a sparse system with $A+\alpha_j M$ and $s+m$ right-hand sides.

Among the algorithms, EKSM has the advantage that the coefficients of the arising linear systems do not change during the iteration. Hence, a single precomputed sparse LU factorization can be reused throughout the whole iteration by means of often significantly cheaper sparse triangular solves. This fact can also be exploited in the presence of an initial feedback K_0, by using the SMW formula once again:

$$y_j = (A - BK_0^*)^{-*} b_j = w_j + g_0(I_m - B^* g_0)^{-1} B^* w_j,$$

where w_j, g_0 solve $A^* w_j = b_j$, $A^* g_0 = K_0$ and also $g_0 \in \mathbb{R}^{n\times m}$ has to be computed only once. Thus, in the unstable situation, except for this extra linear system for g_0, no major additional work is required in EKSM in contrast to the other methods.

For generalized Riccati equations, Table 1 considers only the case when the equivalent CARE defined by $M^{-*}A^*$, $M^{-*}C^*$ is used implicitly in the projection methods, as explained in section 2.1. This means that an extra initial linear solve with M^* is needed in RKSM to compute $M^{-*}C^*$, whereas EKSM requires such an additional solve with M^* in every iteration step. Using sparse Cholesky factors of $M = M^* \succ 0$ would replace a solve with M^* by two sparse triangular solves. Regarding the number of right-hand sides in the occurring linear system, Table 1 reveals that low-rank Newton–Kleinman (NK) iterations have the largest number $(p+m)$ among all algorithms. If the SMW formula is used, linear systems with the matrix $A+\alpha_j M$ and $p+2m$ right-hand sides have to be solved in each iteration step, in contrast to $p+m$ or only p in the other methods. Consequently, among all considered methods, solving linear systems is most expensive in low-rank NK methods. If no initial feedback is given or required, RKSM and ILRSI have an advantage over RADI since they do not need the SMW formula to solve systems with the matrix $A+\alpha_j M$ and p right-hand sides.

3.2. Shifts. With the exception of EKSM, all considered methods require shift parameters to achieve a rapid convergence. Initially, these parameters were computed in advance, prior to any iteration of the Riccati solver. In recent years, dynamic shift generation strategies [17, 24, 48, 49, 72, 101] have attracted increasing attention. In these strategies, the shift needed in a particular iteration step is computed by using all the data that is available to the method in that step. This typically allows the methods to achieve convergence in a smaller number of iterations and with less user interaction, but may increase the computational complexity of each step.

TABLE 1

Form of the occurring linear system in each iteration step. Subscripts $_j$ indicate quantities varying during the iterations. We also provide the number of columns in the right-hand side of the original linear systems and the SMW version.

Method	Coefficients of arising linear system(s)		Columns s of RHS	
	stable (no initial K_0)	with initial K_0	original	SMW
(G)EKSM	M^*, A^*	M^*, $(A - BK_0^*)^*$	p	p
RKSM	$(A + \alpha_j M)^*$	$(A - BK_0^* + \alpha_j M)^*$	p	$p + m$
TRKSM			$p_j \le p$	$p_j + m$
ILRSI			p	$p + m$
RADI	$(A - BK_j^* + \alpha_j M)^*$		p	$p + m$
NK-ADI			$p + m$	$p + 2m$

TABLE 2

Matrices of the arising eigenvalue problems for the dynamic shift generation.

Method	Matrices	Size
RKSM	$\tilde{A}_j$	jp
TRKSM	$\tilde{A}_j$	$\sum_{i=1}^{j} p_i$
ILRSI, RADI	$\mathcal{H}_j^{\text{proj}}$, $\mathcal{M}_j := \text{diag}(U^*MU, U^*M^*U)$	$2\ell p$
NK-ADI	U^*A_kU, U^*MU	$\ell(p + m)$

A substantial amount of work in all the adaptive shift generation strategies considered here is involved in the solution of an eigenvalue problem, whenever new shifts are needed. These eigenvalue problems differ in their size and structure depending on the GCARE method and the particular selection strategy. In Table 2 we summarize the defining matrices of the eigenvalue problems, together with their sizes in the strategies used in this study. As before, j indicates the current iteration number. Although the automatic shift generation of RKSM involves the largest eigenvalue problem, the costs are often still negligible since $jp \ll n$. The dimension of the eigenvalue problem of the approaches in ILRSI, RADI, and NK-ADI depends on a prespecified small number ℓ, for which typical values in the literature are in the range $1 \le \ell \le 6$. In the projection methods (T)RKSM the involved matrices $\tilde{A}_j$ are already constructed as a part of the main iteration loop, while the other methods additionally need to construct them, as well as the orthogonal bases U for certain subspaces just for the purpose of generating the shifts. Note that the approach used in NK-ADI will return more than one shift parameter, so it does not have to be executed in every iteration step.

3.3. Building orthonormal bases of the projection spaces. The projection based methods (G)EKSM and (T)RKSM and the Galerkin accelerated NK-ADI iteration require the construction of orthonormal bases of the used subspaces. Let $Q_{j-1} = [q_1, \ldots, q_{j-1}]$ denote the orthonormal subspace basis after step $j-1$ in either of these methods, and let q_+ denote a block of vectors that need to be added to the subspace in step j. In (G)EKSM, q_+ and each of the q_i have $2p$ columns, while in RKSM they have p columns. In TRKSM, the column dimension of the block q_i is p_i, where $p_i \le p$ is the number of tangential directions that were used in iteration i. We employ the modified blockwise Gram–Schmidt process to expand the basis with q_+:

$$\gamma = q_i^* q_+, \; h_j = h_j + \gamma, \; q_+ = q_+ - q_i\gamma, \; i = 1, \ldots, j-1,$$

and this is done twice (initially, $0 := h_j \in \mathbb{R}^{d_{j-1} \times d_+}$). The new orthonormal basis block q_j is then obtained by a thin QR-factorization $q_+ = q_j h_+$, where $h_+ \in \mathbb{R}^{d_+ \times d_+}$

contains the orthogonalization coefficients. Note that GEKSM uses a different inner product in the orthogonalization process. If complex shifts and directions occur in (T)RKSM, Q_{j-1} is first orthogonally expanded by the real and then by the imaginary parts of the associated complex solution vectors of the linear system. TRKSM in the form proposed in [49] requires the construction of a second orthonormal basis for shift generation and residual norm estimation. In the M-M.E.S.S. implementation [95] of NK-ADI+GP, the Galerkin projection is performed only after a Newton step, and the basis for the projection is orthonormalized using the MATLAB routine `orth`.

3.4. Small-scale CARE solution. The projection based methods (G)EKSM, (T)RKSM, and NK-ADI+GP have to solve small, dense Riccati equations in some stages of the algorithms. Assuming $M = I$ for simplicity, and denoting by $\tilde{A}_j$, $\tilde{B}_j$, $\tilde{C}_j$ the projected matrices A, B, C, respectively, the existence of a stabilizing solution of this small CARE is ensured if $(\tilde{A}_j, \tilde{B}_j)$ and $(\tilde{A}_j^*, \tilde{C}_j^*)$ are stabilizable and detectable, respectively. In [101, Proposition 3.3], a general condition on A and B is given which ensures stabilizability of $(\tilde{A}_j, \tilde{B}_j)$ in the case $M = I$. Unfortunately, in practice it is difficult to check whether this condition holds; in the majority of our experiments the projected CARE could be solved for a stabilizing solution. We observed, however, the occurrence of nearly imaginary eigenvalues of the associated Hamiltonian matrix $\begin{bmatrix} \tilde{A}_j & -\tilde{B}_j\tilde{B}_j^* \\ -\tilde{C}_j^*\tilde{C}_j & -\tilde{A}_j^* \end{bmatrix}$, and these can cause numerical difficulties and accuracy losses in the methods employed for solving the small-scale CAREs (2.1). In such a situation it is reasonable to improve the quality of Y_j by defect correction strategies; see, e.g., [84]. In our experiments we therefore check the quality of Y_j using the norm of $R_j := \tilde{A}_j^* Y_j + Y_j \tilde{A}_j - Y_j \tilde{B}_j \tilde{B}_j^* Y_j + \tilde{C}_j^* \tilde{C}_j$ and, if necessary, try to improve Y_j by running at most two steps of a Newton scheme for the CARE defined by $\tilde{A}_j$, $\tilde{B}_j\tilde{B}_j^*$, R_j.

The cost of solving a dense matrix equation is cubic in the dimension. Since the sizes of the small CAREs arising in each iteration of EKSM and RKSM are equal to the dimension of the projection subspace, the small-scale CARE solution can become expensive in the later iterations. A basic strategy to reduce these costs is to avoid solving the projected matrix equation in each iteration step, and do it only every j_{Gal} steps, where usually we take $j_{\mathrm{Gal}} = 5$. In the Galerkin accelerated NK-ADI iteration, a small-scale GCARE has to be solved only at the end of each outer iteration step.

3.5. Computing or estimating the GCARE residual norm. In Table 3 we summarize the major steps carried out in each method for computing or estimating the GCARE residual norm $\|\mathcal{R}(\tilde{X})\|$. For brevity we only list the parts that involve n-dimensional vectors (matrix vector products with the sparse coefficients A, A^*, M, M^*, inner products of n-vectors, thin QR factorizations, and norms of $n \times x$ block vectors) as those are the most computationally intensive steps in the residual norm calculation. Since the residual norm computation is often tightly intertwined with other parts of the methods (shift generation, solving projected CAREs), the overview only provides computational steps that are exclusively used for that distinct purpose.

The first observation is that in (G)EKSM, RKSM, RADI, and iNK-ADI+LS, the computational effort remains constant during the iteration and is dominated by other operations. That is the reason why the measured computation time for this task is usually negligible. Following the approach in [49, Proposition 6.1], the residual norm computation in TRKSM requires the construction of an additional array of size $n \times \mathbf{s}$, $\mathbf{s} = p + \sum_{j=1}^{j_{\mathrm{it}}} p_j$, leading also to increasing costs. In ILRSI and NK-ADI+GP, where $\|\mathcal{R}(\tilde{X})\|$ is estimated via a Lanczos process applied to $\mathcal{R}$, the effort naturally increases

TABLE 3
Major computational steps involving n-vectors required for GCARE residual norm estimation in all methods.

Method	Computations
(G)E- & RKSM	$2p$ matrix-vector products with M (or factors thereof) and thin QR of $n \times 2p$ matrix
TRKSM	p_j matrix-vector products with M (or factors thereof), additional orthogonalization scheme to expand thin QR of size $n \times \sum_{i=1}^{j-1} p_{i-1} + p$ by p_j vectors
ILRSI	per Lanczos step on $\mathcal{R}$: matrix-vector products with A, A^*, M, M^*, $(2j_{\text{it}} + 1)p + m$ inner products
RADI	norm of $n \times p$ matrix
NK-ADI+GP	per Lanczos step on $\mathcal{R}$: matrix-vector products with A, A^*, M, M^*, $(2j_{\text{it}_{\text{inner}}} + 1)(p + m)$ inner products
iNK-ADI+LS	norm of $n \times (p + 2m)$ matrix

with the iteration because the number of columns in the low-rank factors increases. Note that in NK-ADI+GP, $\|\mathcal{R}(\tilde{X})\|$ only needs to be estimated after each Newton step.

3.6. Memory consumption. Next, we are interested in the memory consumption of the compared methods for both scenarios **S1** and **S2**. For this purpose, we consider the maximum number of n-dimensional vectors stored at any given time during a single run of the algorithm. The memory consumption of other quantities, e.g., the projected matrices, is neglected. We also neglect the memory required for solving the linear systems, since this occurs in all methods and is likely to lead to similar amounts.

We only give some basic estimates on the required number of n-dimensional vectors. These numbers might vary slightly depending on the actual implementation of the algorithms. Assume $M \neq I$ and that only real shifts are used, and let j_{it} denote the total number of iterations required to satisfy the termination criteria in a particular method.

Starting with the projection methods (T)RKSM and (G)EKSM, we recall that they do not gain an advantage in scenario **S2**. The largest number of n-dimensional vectors in both scenarios is used for storing the $j_{\text{it}}p$ basis vectors contained in the matrix $Q_{j_{\text{it}}}$. For RKSM, there are also p auxiliary vectors arising when solving the linear systems, and the block matrix $G_{j_{\text{it}}}$ containing p vectors, which occurs during the residual computation as described in section 2.1. Generating the approximate feedback $K_{j_{\text{it}}}$ requires an additional m vectors. A similar count can be done for block and global EKSM; we only have to replace j_{it} with $2j_{\text{it}}$ since each step of these methods expands the basis twice. For TRKSM, storing the basis requires $\mathbf{s} = p + \sum_{j=1}^{j_{\text{it}}} p_j$ vectors, where p_j denotes the number of direction vectors in the jth iteration. In TRKSM [49, Proposition 6.1], the computation of the shifts, the tangential directions, and the residual norm require additional $\mathbf{s}$ vectors.

The number of n-dimensional vectors in ILRSI can be easily read off from Algorithm 2 and equals $(j_{\text{it}} + 1)p$, along with additional m vectors for the feedback approximation. For the RADI iteration (Algorithm 3) a closer look at the scenarios **S1** and **S2** is appropriate. Starting with **S2**, we require a total of $2p + 2m$ vectors: there are m vectors needed for solving the linear systems via the SMW formula, $2p$ vectors for storing $R_{j_{\text{it}}}$ and V_+, and finally m vectors for storing K_j. In scenario **S1**, the low-rank factor $Z_{j_{\text{it}}}$ naturally adds $j_{\text{it}}p$ vectors. Since we use the residual

TABLE 4
Number of required n-dimensional vectors for all algorithms in scenarios S1 *and* S2*;* $j_{it}{}^{xyz}$ *indicates the executed number of steps of method xyz.*

Method	**S1**	**S2**
(G)EKSM	$2(j_{\text{it}}^{\text{(G)EKSM}}+1)p$	$2(j_{\text{it}}^{\text{(G)EKSM}}+1)p+m$
RKSM	$(j_{\text{it}}^{\text{RKSM}}+2)p$	$(j_{\text{it}}^{\text{RKSM}}+2)p+m$
TRKSM	$2\mathbf{s}+p$	$2\mathbf{s}+p+m$
ILRSI	$(j_{\text{it}}^{\text{ILRSI}}+1+3\ell)p$	$(j_{\text{it}}^{\text{ILRSI}}+1+3\ell)p+m$
RADI	$(j_{\text{it}}^{\text{RADI}}+2+3\ell)p+2m$	$(2+3\ell)p+2m$
NK-ADI+GP	$(j_{\text{it}}^{\text{NK+GP}}+2+3\ell)(p+m)+2m$	$(j_{\text{it}}^{\text{NK+GP}}+2+3\ell)(p+m)+2m$
iNK-ADI+LS	$(j_{\text{it}}^{\text{iNK+LS}}+3+3\ell)(p+m)+2m$	$(3\ell+3)(p+m)+2m$

Hamiltonian shifts in ILRSI/RADI, an additional $3\ell p$ basis vectors are required for storing U_ℓ, A^*U_ℓ, M^*U_ℓ.

Finally, consider the low-rank NK-ADI iteration. Let j_{it} now denote the largest number of inner (ADI) iteration steps reached in any of the outer NK iterations. In both scenarios, the Galerkin projected variant (NK-ADI+GP) requires $j_{\text{it}}(p+m)$ vectors to store the basis, $2p+m$ vectors to use the SMW formula, $2(p+m)$ vectors to store $R_{j_{\text{it}}}$ and V_+, and m vectors to store $K_{j_{\text{it}}}$. In scenario **S1**, the inexact version with line-search (iNK-ADI+LS) needs an additional $p+2m$ vectors to carry out the line-search technique. On the other hand, in scenario **S2**, iNK-ADI+LS does not require all $j_{\text{it}}(p+m)$ vectors to store the low-rank factor $Z_{j_{\text{it}}}$, but only $2\ell(p+m)$ vectors in order to generate the projection based shifts.

These basic estimates for the number of n-dimensional vectors are summarized in Table 4, where superscripts are added to j_{it} to indicate the particular method in question and to highlight that all the methods might need different numbers of steps.

The memory consumption in scenario **S1** is for all methods dominated by the number of required iteration steps j_{it} times the number p. RKSM appears to be efficient in this situation as it only requires a small number of auxiliary n-vectors compared to the other methods. Regarding scenario **S2**, we see that RADI and iNK-ADI+LS have an advantage, since their storage requirements are independent of the number of taken iteration steps j_{it}. The number ℓ is a free parameter that affects the shift generation, and it is typically is chosen very small, e.g., $1 \le \ell \le 6$.

4. Numerical experiments. For all methods described in this paper, in the following comparison we use MATLAB implementations.[1] These incorporate the latest advances of each method, in particular, dynamic shift generation techniques.

State-of-the-art implementations of the low-rank NK iterations can be found in the M-M.E.S.S. package [95], which is a collection of currently mostly low-rank ADI based algorithms for large, sparse GCALEs, GCAREs, differential Lyapunov and Riccati equations, as well as routines for related balancing based model order reduction approaches. Our implementations of the projection based methods of section 2.1 were originally inspired by the source codes available[2] from Valeria Simoncini's homepage. The major differences in our MATLAB codes include a defect correction strategy (cf. section 3.4) for the compressed CAREs and the calculation of the true residual norm for GCAREs as discussed in section 2.1. The linear solves with A in (G)EKSM are

[1]The codes and example data for the experiments in this section are available (see [96]).
[2]http://www.dm.unibo.it/~simoncin/software.html

carried out using sparse LU factors of A, when $A \neq A^*$, and using sparse Cholesky factors of $-A$ when $A = A^* \prec 0$. As described before, these factorizations are computed only once and reused thereafter. In all methods except (G)EKSM, the arising shifted linear sparse systems were solved by the MATLAB backslash "\" operator and by using the SWM formula when needed. If not stated otherwise, the small, dense GCAREs in (G)EKSM and (T)RKSM are solved at every $j_{\mathrm{Gal}} = 5$ iteration steps. The subspace for the shift generation in RADI, IRLSI, as well as NK-ADI is taken from the previous $\ell = 2$ iteration steps, i.e., $U_\ell \in \mathbb{R}^{n\times 2p}$.

The methods were compared on a number of standard examples available from benchmark collections and the literature on large-scale Riccati equations. Our test environment consists of MATLAB 8.0.0.783 (R2012b) running on an Intel Xeon CPU X5650 (2.67GHz) with 48GB RAM. All methods were stopped when

$$\rho := \|\mathcal{R}(\tilde{X})\|_2 / \|C^*C\|_2 \leq \tau_{\mathcal{R}} \text{ with } \tau_{\mathcal{R}} = 10^{-8},$$

where $\tilde{X}$ is the approximate solution produced by the algorithm. Let $\mathcal{L}_k$ indicate the continuous-time Lyapunov operator (2.5a) at iteration k of the NK-ADI method. Then the stopping criterion for the inner Lyapunov ADI iteration is $\|\mathcal{L}_k(\tilde{X})\| < \tau_{\mathcal{L}}\|W^{(0)}\|^2$ with $\tau_{\mathcal{L}} = \tau_{\mathcal{R}}/10$ in the Galerkin accelerated version. In the inexact version it is chosen adaptively via $\tau_{\mathcal{L}_k} = \eta\rho_{k-1}^2$, where $0 < \eta < 1$ and ρ_{k-1} is the GCARE residual norm from the previous Newton step; see [20, 109] for details.

4.1. First test series. For a selection of the examples, Table 5 summarizes the results obtained with the methods introduced in section 2. Next to some basic setup information for the particular example, the table gives the column dimension of the built-up low-rank factor after termination, the numerical rank of the approximate GCARE solution using the machine precision as tolerance, the final scaled GCARE residual ρ_j, the computation times in seconds for subtasks such as solving linear systems (t_{LS}), small-scale GCAREs (t_{care}), computing shift parameters (t_{shift}), the total computation time (t_{total}), the times t_{rest} of all remaining minor computations (e.g., estimating the residual norm and further minor auxiliary routines), and the largest number of stored n-vectors regarding scenarios **S1** (mem_Z) and **S2** (mem_K). The orthogonalization costs for building the orthonormal bases in the projection based methods is included in small-scale solutions times t_{care}. If an algorithm failed for a particular example, a brief comment about the reason is provided. In the following, whenever some defining matrices of a test example are generated randomly, this is to be understood as matrices with normally distributed entries for which the random number generator of MATLAB was at the start of each experiment initialized by `randn('state', 0)`.

We proceed by giving details on the examples and discussing the obtained results for each example.

Example 4.1 (Chip). The *Chip* model [85] represents a finite element discretization of a cooling process of a microchip. This benchmark example from the model order reduction wiki (MorWiki)[3] [106] (also part of the Oberwolfach benchmark collection) provides $A \neq A^*$, $M \succ 0$ diagonal with $n = 20\,082$ and $m = 1$, $p = 5$.

For this first example, all methods generate approximate GCARE solutions of more or less the same rank. EKSM and its global version, GEKSM, achieve the smallest total computation times t_{total}. The reason is clearly that both methods only require a single factorization of A which for this example could be computed

[3] Available at http://modelreduction.org/index.php/Convection

TABLE 5

Testing results. The columns show dimension of the constructed subspace; rank of final low-rank solution; final relative residual norm ρ; time (in seconds) spent solving linear systems t_{LS}, small-scale CAREs t_{care}, and computing shifts t_{shift}; total computation time t_{total}; time t_{rest} for remaining minor subtasks; and peak memory consumption in terms of the number of stored n vectors for scenarios S1 *(mem_Z) and* S2 *(mem_K).*

Ex.	Method	Dim.	Rank	Final res.	t_{LS}	t_{care}	t_{shift}	t_{rest}	t_{total}	mem_Z	mem_K
Chip, $n = 20\,082$, $m = 1$, $p = 5$	EKSM	250	89	1.9e-10	6.2	2.6	–	0.3	9.1	260	261
	RKSM	130	91	1.6e-10	34.6	1.4	2.2	0.5	38.7	140	141
	TRKSM	114	90	5.5e-10	37.8	2.0	0.9	0.3	41.0	238	239
	GEKSM	350	96	7.4e-09	8.0	4.2	–	0.1	12.3	360	361
	ILRSI	180	92	7.2e-10	48.4	–	1.7	3.1	53.3	185	186
	RADI	185	91	2.4e-09	51.5	–	0.7	0.2	52.5	227	42
	NK-ADI+GP	180	92	2.0e-11	45.0	0.6	0.2	1.6	47.4	192	192
	iNK-ADI+LS	180	90	1.7e-09	153.5	–	0.4	1.1	155.0	201	33
Filter3D, $n = 106\,437$, $m = 1$, $p = 5$	EKSM	1 300	218	2.7e-09	422.4	447.4	–	23.4	893.2	1 310	1 311
	RKSM,GEKSM	no convergence in maximum iterations									
	TRKSM	216	210	5.1e-10	138.6	79.6	5.7	5.4	229.3	442	443
	ILRSI	220	210	6.8e-09	107.9	–	17.1	33.3	158.3	255	256
	RADI	215	207	6.0e-09	97.8	–	6.2	2.2	106.2	257	42
	NK-ADI+GP	failure at solving projected CARE									
	iNK-ADI+LS	348	241	1.7e-09	667.7	–	10.8	27.7	706.2	381	51
Rail, $n = 317\,377$, $m = 7$, $p = 6$	EKSM	840	222	3.3e-09	199.3	368.6	–	24.3	592.2	852	859
	RKSM	210	201	3.0e-10	118.3	83.9	11.8	18.3	232.3	222	229
	TRKSM	186	186	7.6e-10	113.7	102.9	3.2	13.1	232.9	384	391
	GEKSM	1800	224	1.6e-09	383.0	2406.1	–	3.7	2792.8	1812	1817
	ILRSI	204	183	5.7e-09	88.6	–	60.7	30.9	180.2	260	267
	RADI	204	183	5.8e-09	101.1	–	27.5	9.5	138.1	264	62
	NK-ADI+GP	324	229	6.2e-14	137.3	42.0	7.4	13.6	200.4	350	350
	iNK-ADI+LS	624	219	2.4e-09	197.0	–	8.3	52.3	257.6	684	86
Toeplitz, $n = 100\,000$, $m = 5$, $p = 20$	EKSM	600	378	1.9e-10	3.5	28.0	–	0.8	32.3	640	650
	RKSM	320	320	8.0e-11	1.3	13.4	0.8	4.8	20.3	365	365
	TRKSM	260	260	6.3e-09	1.0	12.4	2.5	3.7	19.6	305	305
	GEKSM	600	378	1.9e-10	3.3	20.9	–	0.5	24.6	640	650
	ILRSI	340	340	4.1e-09	0.8	–	12.7	15.8	29.3	400	405
	RADI	280	280	4.6e-09	1.0	–	2.9	1.0	4.9	450	170
	NK-ADI+GP	300	300	2.0e-10	0.6	8.0	0.6	1.8	11.0	350	350
	iNK-ADI+LS	350	297	4.7e-09	2.8	–	1.4	6.8	11.0	440	215
Lung, $n = 109\,460$, $m = 10$, $p = 10$	EKSM	100	100	6.6e-09	1.1	1.6	–	0.3	3.0	110	120
	RKSM	120	120	8.4e-10	8.1	3.9	3.0	2.3	17.3	140	150
	TRKSM	failure at solving projected CARE									
	GEKSM	100	93	8.1e-09	1.1	1.4	–	0.1	2.6	110	120
	ILRSI	250	211	9.2e-09	18.2	–	1.4	50.5	70.1	260	270
	RADI	80	80	9.7e-09	7.8	–	1.4	0.4	9.6	180	100
	NK-ADI+GP	failure at solving projected CARE									
	iNK-ADI+LS	440	220	9.7e-09	163.6	–	4.7	14.9	183.2	1 370	190
Stokes, $n = 67\,199$, $m = p = 5$	EKSM	250	104	6.8e-09	8.9	18.2	–	3.1	30.2	260	265
	RKSM	125	125	7.2e-09	15.4	10.5	2.0	4.0	31.9	135	140
	ILRSI	175	106	8.6e-10	19.9	–	2.7	68.4	91.0	180	185
	RADI	110	103	1.5e-09	20.3	–	0.9	0.9	22.1	160	50
	NK-ADI+GP	140	107	4.1e-09	16.1	0.3	0.4	5.8	22.6	160	160
	iNK-ADI+LS	320	100	2.1e-09	69.2	–	2.0	4.2	75.4	365	65

very cheaply and, hence, EKSM and GEKSM also spend a comparatively very small amount of time (t_{LS}) in solving linear systems. However, the generated subspace dimensions are significantly larger compared to all other methods, with GEKSM even further surpassing EKSM. Consequently, a substantial amount (about 50%) of the total computation time is spent on solving the reduced CARE (t_{care}) and the memory

consumptions is higher than for the other methods. This will be a typical observation also in almost all further experiments. Furthermore, there is also a large difference between the subspace dimension and the actual rank of the solution, indicating that the extended Krylov subspace approaches may lead to unnecessarily large projection subspaces for the purpose of solving GCAREs, resulting in a waste of computational effort in several cases, as will be evident in later examples. Using a rational Krylov subspace appears to be a better option, as RKSM and its tangential version, TRKSM, require much smaller subspaces, which are actually close to the rank of the solutions. Thus, smaller storage requirements and small-scale solution times t_{care} can be expected. However, for the Chip example this is compensated for by larger times t_{LS} since the varying linear systems prevent a prefactorization of the system matrix. The setup of the subspaces in TRKSM leads to a further reduction of the subspace dimension compared to the standard block version in RKSM.

Both ILRSI and RADI perform similarly, though slightly worse than (T)RKSM, since they need more iterations. This leads to higher subspace dimensions and linear system and total computation times. Since $m = 1$, the more complicated linear systems in RADI do not cause a significant difference in t_{LS} compared to ILRSI. RADI achieves the smallest memory requirement among all methods in scenario **S2**, which is in line with the discussion in section 3.6.

For the Chip example, the performance of NK-ADI+GP is overall similar to RADI and ILRSI. The Galerkin acceleration led to a termination after the first Newton step. A very poor performance is exhibited by the iNK-ADI+LS iteration, which requires many more linear systems solves and, as a consequence, much higher times t_{LS}, t_{total} compared to the other algorithms. It can only compete regarding the memory consumption in scenario **S2**.

It is also important to emphasize that the shift generation times t_{shift} are always a very small fraction of the total times t_{total} in all methods relying on shift parameters.

Example 4.2 (Filter3D). Another MorWiki [106] example represents a finite element model of a tunable optical filter[4] [28, Chapter 15] with $A \prec 0$, $M \succ 0$, $n = 106\,437$, and $m = 1$, $p = 5$.

Here, EKSM requires far more iterations and therefore a much larger subspace than the other methods. This results in a large amount of time spent in solving the projected CAREs, so that any savings from the easier linear systems solves in EKSM are counterbalanced. RKSM and GEKSM are not able to satisfy the termination criterion in a maximum allowed number of 200 iterations. In view of the subspace dimension, TRKSM, ILRSI, and RADI perform similarly. Because TRKSM has to solve more linear systems with a different coefficient matrix in each step, and a projected CARE every j_{Gal} steps (see section 3.6), its overall computation time is significantly larger than for ILRSI and RADI. The smallest memory consumption in scenario **S2** is again obtained by the RADI iteration. The higher shift generation times t_{shift} in ILRSI compared to RADI are mainly caused by the employed residual Hamiltonian approach requiring to project the residual matrix $\mathcal{R}(X_k)$ onto a low-dimensional subspace. Here, in contrast to RADI, there is no effective low-rank structure of $\mathcal{R}(X_k)$ known for ILRSI. Even though $U_\ell^* \mathcal{R}(X_k) U_\ell$ can be computed without explicitly forming $\mathcal{R}(X_k)$ and by exploiting its symmetry, this construction is, nevertheless, noticeably more expensive than in RADI.

The projected CARE after the first Newton step of the NK-ADI+GP iteration

[4] Available at http://modelreduction.org/index.php/Tunable_Optical_Filter

could not be solved by the employed `care` routine, and the method broke down. The iNK-ADI+LS iteration was able to solve this problem but, as in the Chip example, required larger times t_{LS}, t_{total} caused by the much larger number of encountered linear systems compared to the other algorithms. The dimension of the generated subspace is also higher.

Example 4.3 (Rail). The steel profile cooling models[5] are also part of the MorWiki. They represent spatial finite element discretizations of a two-dimensional heat transfer problem arising in the cooling of steel rail profiles [28, Chapter 19]. Different grid sizes and discretization levels result in five versions of the example, each having different dimension: $n = 1\,357$, $5\,177$, $20\,209$, $79\,841$, $317\,377$.[6] In all versions, $A \prec 0$, $M \succ 0$, and the provided matrices B, C have parameters $m = 7$ and $p = 6$. Table 5 shows the results for the largest version only, while the other versions are examined later.

EKSM, GEKSM, and the iNK-ADI+LS iteration generate substantially larger subspaces than all other algorithms. For EKSM and GEKSM this leads to a very large effort for solving the projected Riccati equation such that, again, any savings gained by the simpler linear systems are completely lost. Apparently, the global method performs far worse than the block method: GEKSM has the largest subspace dimensions and the largest times t_{LS}, t_{care}, t_{total}. These results emphasize again that the extended Krylov subspaces are often not an adequate choice for solving large GCAREs and that, moreover, no computational advantages should be expected by the global over the block approach. Because the iNK-ADI+LS iteration does not rely on solving projected GCAREs, its overall computation times are significantly smaller compared to (G)EKSM, but clearly larger than for (T)RKSM, ILRSI, and RADI. The latter four algorithms appear to be the winners for this example, having much smaller subspace dimensions that are, especially for (T)RKSM, very close to the actual solution rank. ILRSI and RADI need the smallest computation times, overall, since no projected GCARE has to be solved. The NK-ADI+GP iteration is able to compete for this example, with total computation times in between those of (T)RKSM and ILRSI, RADI, but with a significantly smaller final residual norm. This is due to the fact that NK-ADI+GP generates a roughly 1.5 times larger subspace compared to (T)RKSM and ILRSI, RADI, such that the Galerkin projection results in a much smaller residual norm than actually desired.

Example 4.4 (Toeplitz). This is an artificial example defined by the Toeplitz matrix

$$A = \begin{bmatrix} 2.8 & 1 & 1 & 1 & 0 & \dots & \\ -1 & 2.8 & 1 & 1 & 1 & 0 & \ddots \\ 0 & -1 & 2.8 & 1 & 1 & 1 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & & & 0 & -1 & 2.8 \end{bmatrix}, \quad M = I_n,$$

and random B, C with $\|B\| = 1$. We set $n = 100\,000$, $m = 5$, and $p = 20$. The idea for this setting comes from [81, Example 7.3], [102, Example 5.5], where slightly different entries and $m = p = 1$ are used. We use this example because the arising linear systems are extremely cheap to solve even for large n, which helps to emphasize the required work in other stages of the algorithms.

[5] Available at http://modelreduction.org/index.php/Steel_Profile

[6] The largest version was obtained by a FEniCS based reimplementation (available at http://gitlab.mpi-magdeburg.mpg.de/models/fenicsrail/) of the discretization.

The data given in Table 5 confirm this because the timings t_{LS} represent only a small fraction of t_{total} for all methods. The timings t_{care}, and in some cases even the shift generation times t_{shift}, are higher than t_{LS}. In fact, more than 50–75% of time is spent in solving the small-scale CAREs. The methods (G)EKSM again build up larger subspaces than the other methods. All of the other approaches end up with significantly smaller subspace dimensions, which are all close to the actual solution rank. The approximate solution of the smallest rank is obtained by TRKSM, which, having to solve projected CAREs, in the end does not achieve the smallest computation times t_{total}. Because the RADI and iNK-ADI+LS methods do not have to solve such projected CAREs, they achieve the smallest total times t_{total} with a substantial margin. On par with iNK-ADI+LS regarding t_{total} is the NK-ADI+GP method, which only requires one small CARE solve after the first Newton step. The cost for this is, however, a substantial portion of the overall cost, as reflected by t_{care}. Although ILRSI does not work with projected CAREs either, it ends up with the highest total time t_{total} for two reasons: first, the shift generation is more costly (see observations made for Filter3D), and second, the costs for the residual norm estimation via a Lanczos process in this example are comparatively more expensive (around 50% of the total computation time). Note that without the scaling $\|B\| = 1$, ILRSI converges much more slowly because it encounters problems in generating good shift parameters.

For this example, we also tried SDA [80], but it was not able to compute a low-rank approximate solution of the desired accuracy. We terminated SDA after 5 iteration steps which already took 247.8 seconds and led to a relative CARE residual norm $\|\mathcal{R}(\tilde{X})\|_2/\|C^*C\|_2 \approx 0.998$. Interestingly, the built-in residual norm estimation of SDA delivered a value $2.53 \cdot 10^{-13}$ and incorrectly indicated convergence of the method. The produced low-rank solution factors after these 5 iteration steps had reached the maximal allowed column dimension 600.

Example 4.5 (Lung). The example LUNG2 from the SuiteSparse Matrix Collection [42] (formerly the University of Florida Sparse Matrix Collection) models temperature and water vapor transport in the human lung. It provides matrices with leading dimension $n = 109\,460$, where A is nonsymmetric, $M = I$, and B, C are generated as random matrices with $m = p = 10$.

For this example, (G)EKSM wins regarding the total computation times t_{total} since, in contrast to the prior examples, they do not produce larger subspace dimension than the other methods. The extended Krylov method appears to be a viable choice for this example, and the cheaper linear solves, similar to the Chip example, pay off here. RKSM has a larger time t_{LS}, because it, again, has to put more effort into solving the varying linear systems. TRKSM and NK-ADI+GP break down because the employed routine `care` for solving the projected CARE fails at some point. RADI achieves the smallest subspace dimension, but needs more total time t_{total}, due to the varying linear systems. For this example, ILRSI encounters problems in the dynamic shift generation and requires many more iterations compared to RADI. We expect these issues to be the result of complex data in the basis used for projecting the residual Hamiltonian matrix. Similar to prior observations, iNK-ADI+LS takes by far the last place regarding several measures, e.g., subspace dimension, t_{LS}, and t_{total}.

Example 4.6 (Stokes). A spatial discretization of a two-dimensional Stokes equation from [98] is included as test case of M-M.E.S.S. consisting of matrices

$$A = \left[\begin{smallmatrix} A_1 & G \\ G^* & 0 \end{smallmatrix}\right], \quad M := \left[\begin{smallmatrix} M_1 & 0 \\ 0 & 0 \end{smallmatrix}\right] \in \mathbb{R}^{n\times n}, \quad B = \left[\begin{smallmatrix} B_1 \\ 0 \end{smallmatrix}\right] \in \mathbb{R}^{n\times m}, \quad C = \left[\begin{smallmatrix} C_1 & 0 \end{smallmatrix}\right] \in \mathbb{R}^{p\times n},$$

with $n = n_1 + n_2$, $n_1 > n_2$, $A_1, M_1 \in \mathbb{R}^{n_1 \times n_1}$ sparse, $G \in \mathbb{R}^{n_1 \times n_2}$, $B_1 \in \mathbb{R}^{n_1 \times m}$, $C_1 \in \mathbb{R}^{p \times n_1}$. This results in a descriptor system of index 2, such that using a Riccati based feedback stabilization requires some additional steps. For the sake of brevity, we only give the absolutely necessary information regarding the handling of those structured descriptor systems, especially with respect to the numerical steps in the GCARE algorithms. With $\operatorname{rank}(G) = n_2$ we can associate the projector

$$\Pi := I_{n_1} - G(G^* M_1 G)^{-1} G^* M_1$$

to the differential algebraic system. The purpose of Π is to ensure that algebraic constraints of the descriptor system are satisfied, i.e., that the calculations happen in the correct hidden manifold. Following [10, 11, 21, 60, 109], the GCARE to be solved is defined by the large, dense matrices $\mathcal{A} = \Theta_1^* A_1 \Theta_1$, $\mathcal{M} = \Theta_1^* M_1 \Theta_1$, $\mathcal{B} = \Theta_1^* B_1$, $\mathcal{C} = C_1 \Theta_1$, where $\Theta_1, \Theta_2 \in \mathbb{R}^{n_1 \times (n_1 - n_2)}$ are factors of Π: $\Pi = \Theta_1 \Theta_2^* \in \mathbb{R}^{n_1 \times n_1}$.

It has been shown in [11, 21, 109] that the main numerical subtasks in GCARE methods based on the low-rank NK-ADI framework can be implemented without the explicit projection Π or its factors Θ_1, Θ_2, such that working with the original matrices A, M, B is sufficient and numerically desirable since A, M are sparse. Only the transformed right-hand side factor $C_\Pi := (\Pi C_1^*)^*$ is required to start the iteration. It is straightforward to carry these ideas over to the direct iterations (section 2.2) and the projection methods (section 2.1). An application of the projection Π requires solving a symmetric indefinite linear system in saddle point form defined by $\hat{M} := \left[\begin{smallmatrix} M_1 & F \\ F^* & 0 \end{smallmatrix}\right]$. In the projection based methods, it is possible that the algorithms drift off the hidden manifold [104], especially after the orthogonalization scheme for expanding the orthonormal basis. Hence, it is wise to apply Π also to the outcome of the orthogonalization routine (repeated modified Gram–Schmidt in this exposition). Moreover, estimating the GCARE residual norm via a Lanczos process, as it is done in ILRSI, NK-ADI+GP, also requires applications of Π in each Lanczos step [109, Chapter 4.3]. Consequently, the occurrence of Π-application adds an additional source of numerical cost to each method not present in the examples before. How much extra effort this introduces depends on how often Π needs to be applied: once in RADI, iNK+ADI+LS, $1 + j_{\text{it}}$ times in EKSM, (T)RKSM, j_{Lan} times after each Galerkin projection in NK-ADI+GP, and $1 + j_{\text{it}} j_{\text{Lan}}$ times in ILRSI, where j_{Lan} indicates the number of executed Lanczos steps for estimating the GCARE residual norm. The computation of the residual Hamiltonian shifts in ILRSI also relies explicitly on $\mathcal{R}$, s.t. further Π-applications are needed (cf. discussion in Example 4.2).

Here, we use a discretization with 150 grid points in each spatial dimension resulting in $n = 67\,199$, $n_1 = 44\,700$, $n_2 = 22\,201$, $m = p = 5$. These sizes still allow us to employ sparse-direct techniques to handle the linear systems defined by $\hat{M}$ as well as all further occurring linear systems in the GCARE methods. In particular, we use a sparse LDL^* factorization of $\hat{M}$ which is computed once before the iterations.

The results are given at the bottom of Table 5. The winner for this example in terms of computation time, subspace dimension, and memory requirements is RADI, closely followed by the projection methods EKSM, RKSM, and NK-ADI+GP, although EKSM again produced larger subspaces. Similarly to before, iNK+ADI+LS required many more LR-ADI steps leading to larger times t_{LS}, t_{total} and larger subspace dimensions. The last place takes ILRSI because of the significantly more expensive residual norm estimations leading to large t_{rest} timings. This is similar to the *lung* example, but here, the Lanczos process additionally requires Π-applications.

This concludes this first series of examples. We point out that the amount of

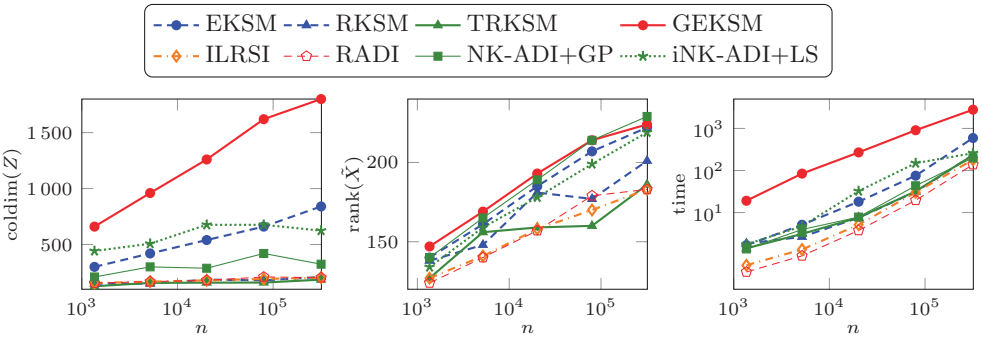

FIG. 1. *Scaling with respect to the problem dimension of Rail examples: Plotted are the generated subspace dimension (left), the ranks of the approximate solutions (middle), and the total computation time (right) versus the leading dimension n.*

work for adaptively generating shifts is, with minor exceptions, only a small fraction of the overall computational effort due to the advances in this area [17, 24, 48, 49, 72, 101] in recent years. If cleverly implemented, precomputed shifts could also be generated efficiently, but in our experience, the performance of the low-rank GCARE (and GCALE) methods typically lags behind compared to dynamic shift selection.

Although not the topic of this study, we expect a similar conclusion regarding the related methods for large, sparse Lyapunov and Sylvester equations. We did not put timings for the residual norm computation in Table 5 because these were only a tiny fraction of the overall computation time, mostly because of low-rank or otherwise exploitable structures of the GCARE residual matrix which made this task significantly more manageable than with earlier approaches. The clear exception is here ILRSI, which still relies on a Lanczos process for this task and, hence, could benefit from low-rank expressions with fixed-sized factors of the GCARE residual. This would further improve the performance of the employed residual Hamiltonian shift generation in ILRSI.

4.2. Scaling of the performance with respect to the problem dimensions. Now we examine the behavior of the algorithms when the leading dimension n is increased. For this, we use all five versions, described above, for the Rail example, which correspond to different coarseness of finite element meshes.

The subspace dimensions, the solution ranks, and total computation times are plotted against the different values of n in Figure 1. For this particular example, the increasing dimensions do seem to only lead to moderately larger ranks of the approximate GCARE solutions required to satisfy the stopping criterion; see the middle plot of Figure 1. Hence, if the algorithms manage to produce a subspace of dimension not far from the solution rank, the numerical effort should essentially only increase with n because the linear systems are increasingly expensive to solve. In the left plot of Figure 1 we see that, indeed, most methods end up with subspace dimensions that remain approximately unchanged for increasing n. The striking exceptions are (G)EKSM, whose subspace dimensions are again much larger than those of the other methods and also clearly increase with n. Similar to prior observation, GEKSM performs worse than EKSM. For the largest example, GEKSM requires a subspace dimension more than three times as large as for the other methods. The NK-ADI

methods produce somewhat larger and slightly increasing subspace dimensions compared to (T)RKSM, ILRSI, and RADI.

The total computation times illustrated in the right plot of Figure 1 indicate that, as expected, increasing system dimensions n lead to increasing computation times because of the larger linear systems. The trend in which t_{total} increases is similar for (T)RKSM, ILRSI, RADI, NK-ADI+GP, especially for dimensions $n > 10^4$. The timings for (G)EKSM and the iNK-ADI+LS iteration are somewhat larger than for the other algorithms. For the iNK-ADI+LS iteration, the difference in t_{total} compared to the other methods except GEKSM appears to decrease for increasing n.

The purpose of the next examples is to study how the change in the dimensions m and p affects the performance of the algorithms. We keep the matrices A and M fixed but alter the number of columns in B and the number of rows in C. For most methods, this will result in a different number of columns in the right-hand side when solving linear systems; see Table 1. Furthermore, for the projection based methods, the size of p also dictates the growth of the subspace dimension and, thus, significantly influences the cost for solving the Galerkin systems. Another way of manipulating the influence of linear systems during the algorithm runs is to provide an initial guess X_0, e.g., when solving an unstable Riccati equation. Changing m changes in this case the number of right-hand sides in the linear system if the SMW formula is employed (cf. Table 1); we study this effect as well.

Example 4.7 (CUBE-FD). Consider a centered finite difference discretization of the differential equation

$$\partial_t f(\xi,t) = \Delta f(\xi,t) - 10\xi_1 \partial_{\xi_1} f(\xi,t) - 1\,000\xi_2 \partial_{\xi_2} f(\xi,t) - 10\partial_{\xi_3} f(\xi,t) + b(\xi)u(t),$$

on a unit cube with $\xi = (\xi_1, \xi_2, \xi_3)$. Using Dirichlet boundary conditions and $n_0 = 32$ nodes in each spatial direction yields a nonsymmetric matrix A of order $n = n_0^3$ and $M = I_n$. The matrices $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ are generated randomly. For $n_0 = 22$, $C = B^*$, and $m = 10$, this is exactly [100, Example 5.2].

We will also use the last example to evaluate the influence of a nonzero initial feedback K_0. However, for several unstable variations of this example (e.g., using shifted matrices $A + \psi I$, $\psi > |\max \mathrm{Re}(\lambda(A))|$), we encountered severe problems with the initial feedback generation explained in section 2.5. Either the required eigenvectors or Schur vectors could not be computed in a stable manner, or the small matrix equations (Bernoulli or Lyapunov) could not be solved satisfactorily. To avoid these problems, we modify Example 4.7 as follows.

Example 4.8 (CUBE-FD-unstable). Consider $A_u = \mathrm{diag}(A, A_+)$, $B_u^* = [B^*, B_+^*]$, where A, B are as in Example 4.7, $A_+ \in \mathbb{R}^{u\times u}$ is an artificially generated matrix with eigenvalues in $\mathbb{C}_+$, and $B_+ \in \mathbb{R}^{u\times m}$, $C \in \mathbb{R}^{p\times u+n}$ are given and randomly generated. We used $u = 5$ in our experiments. The required antistable invariant subspace for the above A_u is trivially spanned by $Q_u = [0, I_u]^* \in \mathbb{R}^{(n+u)\times u}$ and it holds that $Q_u^* B_u = B_+$, so we set $K_0 = [0, B_+^*]^*$. To generate the antistable part A_+, we enforce the solution of the Bernoulli equation $A_+ S_u + S_u A_+^* - S_u (Q_u^* B)(B^* Q_u) S_u = 0$ to be $S_u = I_u$. Therefore, $A_+ + A_+^* = B_+ B_+^*$, and we simply take $A_+ = \frac{1}{2} B_+ B_+^*$. We again point out that this construction of an unstable system is done entirely for the purposes of demonstrating the effects of an initial feedback K_0. From a practical point of view, the way A_u, B_u, C_u are built makes this setting viable for partial stabilization approaches [14, 58, 107]. Moreover, as we mentioned earlier, projection based methods might, under certain conditions [101, Proposition 3.3], not need an initial guess to converge.

TABLE 6

Testing results for CUBE-FD and CUBE-FD-unstable with $n = 32\,768$, $\tau_{\mathcal{R}} = 10^{-8}$, and different values of m, p.

Setup	Method	Dim.	Rank	Final res.	t_{LS}	t_{care}	t_{shift}	t_{rest}	t_{total}	mem_Z	mem_K
$m = 10$, $p = 10$	EKSM	1 200	568	6.9e-09	27.7	142.7	–	1.9	172.4	1 220	1 230
	RKSM	460	460	7.4e-09	114.3	17.3	6.3	2.6	140.4	480	490
	TRKSM	470	470	5.4e-09	114.4	18.4	26.7	1.9	161.5	960	970
	GEKSM	1 300	558	5.7e-09	29.8	157.8	–	0.3	187.9	1 320	1 330
	ILRSI	520	490	9.6e-09	142.5	–	18.6	19.5	180.6	570	580
	RADI	520	471	8.9e-09	186.8	–	5.8	1.1	193.6	620	100
	NK-ADI+GP	550	550	1.5e-10	134.7	9.3	0.7	1.1	145.8	590	590
	iNK-ADI+LS	1 120	546	9.4e-09	477.6	–	3.3	3.3	484.9	1 210	190
$m = 10$, $p = 20$	EKSM	2 400	1 099	4.4e-09	56.8	944.3	–	4.1	1 005.3	2 440	2 450
	RKSM	1 120	1 100	3.2e-10	200.4	114.7	24.9	6.7	346.6	1 160	1 170
	TRKSM	1 060	1 060	4.6e-10	185.0	92.7	131.4	5.3	414.5	2 160	2 170
	GEKSM	2 800	1 096	3.0e-09	67.8	1 553.4	–	0.6	1 621.8	2 840	2 850
	ILRSI	1 040	978	6.0e-09	199.4	–	52.2	39.8	291.3	1 140	1 150
	RADI	1 040	942	7.0e-09	246.6	–	18.7	2.3	267.7	1 220	180
	NK-ADI+GP	1 140	1 120	4.0e-11	204.9	50.5	2.0	2.3	259.6	1 200	1 200
	iNK-ADI+LS	1 740	1 081	6.6e-09	616.6	–	6.6	6.6	629.4	1 860	270
$m = 20$, $p = 10$	EKSM	1 200	568	8.2e-09	25.6	148.0	–	1.9	175.5	1 220	1 240
	RKSM	460	460	7.7e-09	126.9	18.6	6.7	2.6	154.8	480	500
	TRKSM	480	480	6.0e-09	129.5	21.5	29.3	2.0	182.3	980	1 000
	GEKSM	1 400	569	2.8e-09	29.1	210.6	–	0.2	239.9	1 420	1 240
	ILRSI	530	509	7.3e-09	173.6	–	20.5	22.4	216.5	580	600
	RADI	540	499	2.1e-09	269.4	–	8.2	1.5	279.1	660	120
	NK-ADI+GP	570	570	6.2e-11	156.2	10.1	0.7	1.2	168.2	630	630
	iNK-ADI+LS	1 680	534	9.7e-09	692.1	–	5.8	16.0	703.9	1 830	300
$m = 20$, $p = 10$ unstable	EKSM	1 200	571	8.0e-09	29.4	139.2	–	0.4	169.0	1 220	1 240
	RKSM	570	545	2.5e-09	168.5	28.1	11.0	3.3	210.9	590	610
	TRKSM	464	464	5.5e-09	136.7	21.9	50.6	2.1	211.3	948	968
	ILRSI	600	504	5.5e-09	187.7	–	4.9	108.8	301.4	670	680
	RADI	600	490	5.5e-09	280.5	–	4.9	1.8	287.2	720	120
	NK-ADI+GP	800	572	9.2e-11	267.3	37.5	1.9	4.8	311.5	860	860
	iNK-ADI+LS	2 190	585	9.0e-09	1 467.1	–	11.8	15.5	1 494.4	2 550	300

Table 6 and Figure 2 show the results. We first comment on the top row in Figure 2; the number m of columns of B is fixed to 10, while the number p of rows in C increases. The generated subspace dimensions as well as the obtained solution ranks increase almost linearly with increasing p for all methods. This is expected, as each step in each method (except TRKSM) expands this subspace by a multiple of p vectors. The slope in the top left graph of Figure 2 is the largest for (G)EKSM, since these two methods add $2p$ vectors in each step, while the rest add only p vectors. The iNK-ADI+LS method appears to build subspaces larger than the rest, with the exception of (G)EKSM. The linear increase in t_{LS} for all methods is obvious, and for the methods not based on projection, the total time of computation increases linearly as well. However, in the projection methods, the costs of solving larger projected GCAREs are getting increasingly expensive as p gets larger, resulting in weaker performance. This is, again, most obvious for (G)EKSM.

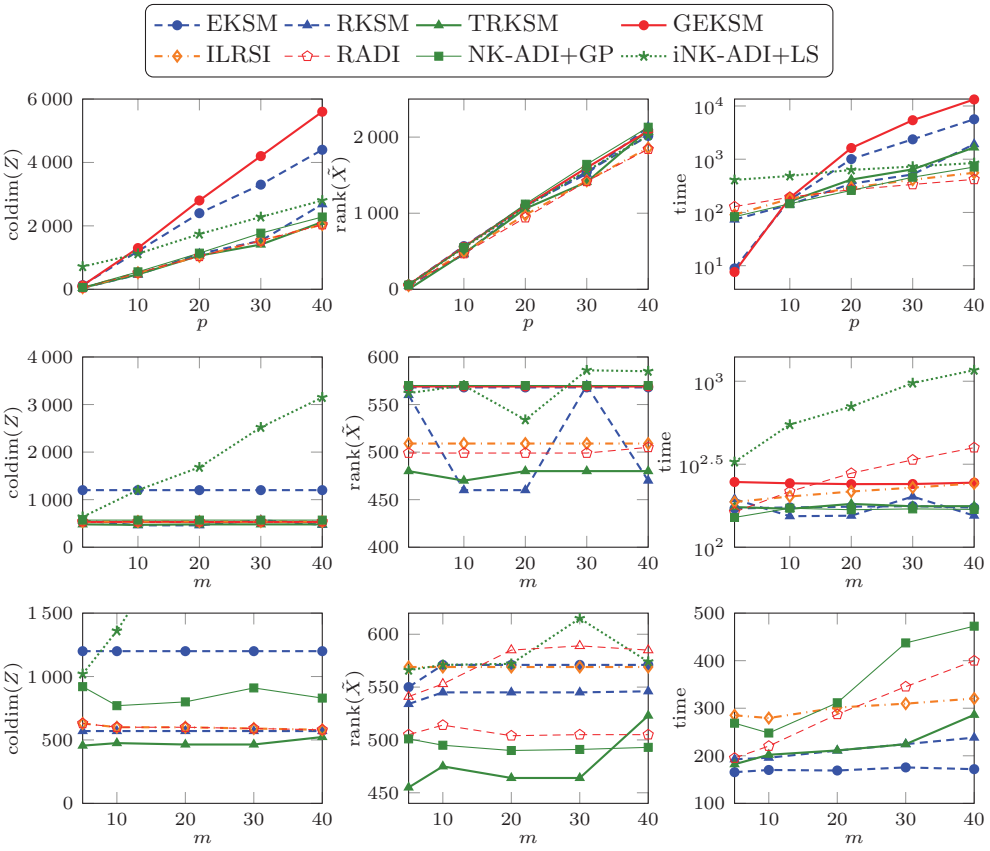

FIG. 2. *CUBE-FD(-unstable): Results with respect to different values of m and p: Generated subspace dimension (left), the ranks of the approximate solutions (middle), and the total computation time (right) versus p or m. The top row of plots refers to $m = 10$, $p = 1, 10, 20, 30, 40$, the middle row refers to $p = 10$, $m = 1, 10, 20, 30, 40$, and in the bottom row are the results for the unstable case CUBE-FD-unstable with $p = 10$, and varying $m = 5, 10, 20, 30, 40$ are plotted.*

Next, we study the middle row of Figure 2: now $p = 10$, while m increases. Apparently, this increase has no effect on the generated subspace dimension at all, which is clear from the way the subspaces are expanded and, thus, indicates that the iteration numbers also remain approximately constant. The exception is iNK-ADI+LS, where after the first Newton step, $p + m$ vectors are added in each inner iteration. However, the obtained solution ranks appear to be more sensitive regarding different values of m, especially for RKSM, iNK-ADI+LS. The effect on the total time is also different: the methods that use the SMW formula for the solution of linear systems (RADI, iNK-ADI+LS) see a linear increase in total time, which is a consequence of the increase in t_{LS}. The NK-ADI+GP is for this setting not affected by larger B, because in all cases the Galerkin acceleration led to satisfaction of the stopping criterion after the first Newton step. Consequently, each encountered linear system had only p right-hand sides, and the SMW formula was not necessary. The timings for ILRSI increase slightly since, for reasons similar to those outlined above, the shift generation became more expensive with increasing m.

Finally, we analyze the case of unstable Riccati equations for which we fix $p = 10$ and vary $m = 5, 10, 20, \ldots, 40$. The bottom row in Figure 2 shows the results. For this setting, GEKSM does not produce reasonable results (stagnation at large residual norms or problems while solving the projected CARE) and is hence omitted. ILRSI encounters problems in the shift generation routine, and thus we use it with the shifts generated by RADI. The related timings t_{shift} of ILRSI are copied from those of RADI. Despite the nonzero initial feedback, larger values of m do not appear to severely affect the final subspace dimensions for all methods. The strong exception is the iNK-ADI+LS iteration, which again performs comparatively badly for this example, similar to the cases described above. Only for this method, the subspace dimensions are much larger than for the other methods, surpassing even EKSM, and increasing m clearly seems to lead to larger dimensions as well. We cut off the associated bottom left plot in Figure 2 at 1 500 because otherwise the very large values of iNK-ADI+LS distort the whole plot. Some variations in the solution ranks can also be seen for varying m, similar to the previous experiment with stable matrix pair (A, M). The total computation times t_{total} increase at various rates for different methods as m increases, which is due to the increase in the time t_{LS} needed for solving linear systems, since all of the methods use the SMW formula now. As explained in section 3.1, EKSM has an advantage here, and its computation times only increase marginally. RADI and NK-ADI+GP exhibit the strongest increase of t_{total}. The same holds for iNK-ADI+LS whose curve is omitted in the bottom right plot in Figure 2.

For $m = 20$, the results are listed in the bottom section of Table 6. We see that EKSM is the fastest, followed by (T)RKSM; these two methods achieve the smallest subspace dimensions. It is also evident that iNK-ADI+LS cannot keep up with any of the other algorithms. Note that the residual norm estimation via Lanczos in ILRSI takes up a significant portion of the overall time here.

In respect to the memory consumptions for scenario **S2**, RADI clearly requires the smallest number of n-vectors to be stored. The NK-ADI+GP achieves, similarly to the *rail* example before, smaller final residual norms, which is again a result of the larger generated subspace dimensions.

4.3. Comparison of theoretically equivalent methods. As a final experiment, we test to what extent the predicted equivalence [17, Theorem 2] of ILRSI [81, 83] and RADI [17] holds in practice. At first, both methods are executed with the same set of predetermined shift parameters. We mimic this by running RADI with the adaptive residual-Hamiltonian shift strategy and then use the generated shifts within ILRSI. In the second experiment, we let ILRSI compute its own shifts using the residual Hamiltonian approach, too. The *Rail* example with $n = 79\,841$ and the *CUBE-FD* example with $n = 10\,648$, $m = p = 10$ are used for this study. Figure 3 shows the scaled residual norms $\rho_j := \|\mathcal{R}_j\|/\|CC^*\|$ for RADI and ILRSI with precomputed and adaptive shifts, as well as the norm differences $\Delta\rho_j := |\|\mathcal{R}_j^{\text{RADI}}\| - \|\mathcal{R}_j^{\text{ILRSI}}\||/\|CC^*\|$ against the iteration number j.

In both cases, we indeed observe that RADI and ILRSI produce visually nearly indistinguishable residual curves when using exactly the same shifts. When ILRSI generates its own shifts for the *CUBE-FD* example, the discrepancies $\Delta\rho_j$ are larger but still small. The reason is that for this example, some of the generated shifts come in complex conjugated pairs, but in its current implementation, ILRSI cannot handle those complex shifts similarly well as the RADI method. At some point, the low-rank solution factors generated by ILRSI will be complex, which will lead to small differences in the computed shifts since parts of the low-rank factor are used to project

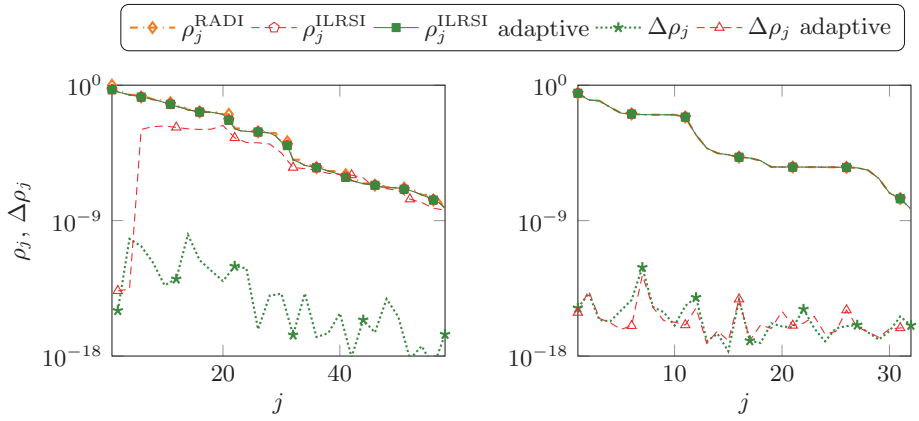

FIG. 3. *Comparison of RADI and ILRSI with and without adaptive shifts for the CUBE-FD (left) and rail (right) examples: Scaled residual norms ρ_j and discrepancies $\Delta\rho_j$ against iteration number j.*

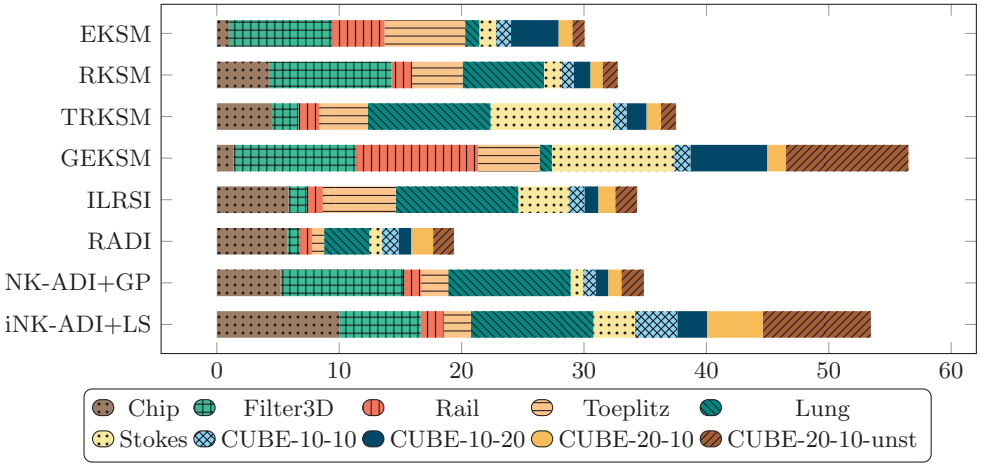

FIG. 4. *A summary of the performance of all methods over all examples. For each example, a method gains a penalty equal to its running time divided by the running time of the fastest method for that example. If the quotient is larger than* 10*, or the method failed in that example, the penalty is set to* 10*. The smaller the total penalty, the faster the method.*

the Hamiltonian matrices. Similar observations can be made for the *Rail* example, but the norm differences $\Delta\rho_j$ in case of adaptive shifts are much smaller compared to the *CUBE-FD* example as the majority of generated shifts for this example is real. In all cases, the differences with respect to the computed approximate Riccati solutions $\|X_j^{\text{RADI}} - X_j^{\text{ILRSI}}\|$ show the same behavior as the residual norm differences.

5. Conclusions. In this paper, we studied three classes of low-rank algorithms for finding approximate solutions of large-scale GCAREs: the projection based extended and rational Krylov subspace methods (EKSM and RKSM), direct iterations not bound to a projection framework (ILRSI and RADI), and Newton–Kleinman methods. Some modifications of the considered methods were also taken into ac-

count. We discussed the methods with respect to the computational complexity of solving the matrix equation, and the amount of memory required. The theoretical analysis was confirmed by a number of numerical experiments.

We detected the most important subtasks that influence the computational work: solving linear systems with multiple right-hand sides, solving small, dense GCAREs, and generating shift parameters. The only subtask present in each iteration of every method is the solution of linear systems, where the number of columns in the right-hand sides varies from one method to the other. This leads to different behavior regarding the ranks of the constant and quadratic term defining the GCARE. In all methods which do not rely on solving small, dense GCAREs (ILRSI, RADI, iNK-ADI+LS), these linear solves constitute the largest portion of the overall computational work. For the projection based methods ((G)EKSM, (T)RKSM, NK-ADI+GP), the solution of the dense matrix equations could, depending on p, m, and the required number of iteration steps, be a second substantial portion of the work load. The generation of shift parameters and the estimation of the GCARE residual norms are, due to several recent advances, often only minor fractions of the total computational cost.

Projecting to rational Krylov subspaces often fared better than projecting to extended Krylov subspaces, typically keeping the built-up subspace dimension close to the actual rank of the computed low-rank solution. The extended Krylov subspace methods tend to generate subspaces of dimensions considerably larger than the rank of the approximate solution, and this results in a significant increase of the effort to solve the projected dense GCAREs. The direct iterative methods, ILRSI and RADI, are theoretically equivalent, which can also be observed in numerical tests where only smaller deviations occur due to round off. In the present form, ILRSI still lacks some efficiency improvements found in the other methods, like reducing the occurrence of complex arithmetic operations and a cheap GCARE residual norm estimation. Hence, ILRSI could sometimes not keep up with RADI or some of the other approaches in our experiments. If these issues can be solved in future research efforts, ILRSI can potentially become a very competitive approach. The low-rank Newton–Kleinman iterations showed a mixed performance. While the Galerkin projection version could compete most of the time (provided the reduced GCARE could be solved), the inexact version with line search could not keep up in most cases. The reasons were mostly the more expensive linear systems, due to a higher number of right-hand sides, and the substantially larger number of iteration steps required.

Another considered performance indicator was the memory consumption, where it is important to distinguish between two scenarios. The first scenario occurs when an approximate low-rank solution $\tilde{X} \in \mathbb{R}^{n\times n}$ of the GCAREs is sought. Therefore, the smaller the constructed subspace or low-rank solution factor the better. In this context, the (T)RKSM provided especially good results, followed by RADI and ILRSI. At the expense of somewhat larger generated subspace dimensions, NK-ADI+GP achieved sometimes several orders of magnitude smaller residual norms. In the second scenario, only a stabilizing feedback matrix $\tilde{K} = M^*\tilde{X}B \in \mathbb{R}^{n\times m}$ is required, naturally asking for less data to be computed. Obviously, projection based methods do not gain an advantage here, since they still have to build the basis for the entire subspace. The same is true for ILRSI. Only RADI and iNK-ADI+LS were able to solely operate on approximate feedback matrices without ever forming the low-rank solution factors, making them more memory efficient in this situation.

Averaging over all carried out tests, (T)RKSM and RADI yielded smaller subspace dimensions (low-rank solution factors) as well as smaller total computation

times (see Figure 4) compared to the other approaches. The ultimate choice between (T)RKSM and RADI should then take into account whether the full low-rank solution or only a stabilizing feedback is sought.

We also experimented with unstable GCAREs, which might ask for an initial stabilizing guess. The generation of such an initial guess proved to be demanding in several tests because the conventional approach used for that purpose often failed. This issue requires further research effort.

In this study we restricted ourselves to sparse direct solution techniques for the arising linear systems of equations. In general, preconditioned iterative solvers could also be applied. This points towards the interesting research topic of investigating the effects of the errors made by solving the linear systems only inexactly, and to establish rules on the minimal accuracy of the linear solves required to obtain a low-rank GCARE solution of a certain quality.

Similar further comparative studies could address low-rank methods for continuous-time Lyapunov equations, and also discrete-time Lyapunov and Riccati equations, as well as nonsymmetric variants. A comparison of different low-rank approaches for differential matrix equations is ongoing work. Especially in the latter, but also in other applications, the inhomogeneities of the arising GCAREs can be indefinite, which requires some smaller changes in the implementations of the algorithms; see, e.g., [71, 75]. A more demanding alteration is when the quadratic term in the GCARE is positive semidefinite (i.e., M^*XBB^*XM occurs with a positive sign in (1.1)). This arises, e.g., in certain model order reduction approaches, for which some numerical methods are proposed in [33].

Acknowledgments. We thank Valeria Simoncini, Tatjana Stykel, Arash Massoudi, Peter Chang-Yi Weng, and Heiko Weichelt for helpful discussions and for sharing their implementations of (T)RKSM, GEKSM, ILRSI, SDA, and iNK-ADI+LS. The latter especially were a great asset for developing the implementations used in this study. Additional thanks go to Tony Stillfjord for his careful revision of our final draft. This work was primarily generated while Patrick Kürschner was affiliated with the Max Planck Institute for Dynamics of Complex Technical Systems in Magdeburg, Germany.

REFERENCES

[1] E. L. Allgower and K. Georg, *Introduction to Numerical Continuation Methods*, Classics Appl. Math. 45, SIAM, Philadelphia, 2003, https://doi.org/10.1137/1.9780898719154.

[2] L. Amodei and J.-M. Buchot, *An invariant subspace method for large-scale algebraic Riccati equation*, Appl. Numer. Math., 60 (2010), pp. 1067–1082, https://doi.org/10.1016/j.apnum.2009.09.006.

[3] L. Amodei and J.-M. Buchot, *A stabilization algorithm of the Navier—Stokes equations based on algebraic Bernoulli equation*, Numer. Linear Alg. Appl., 19 (2012), pp. 700–727, https://doi.org/10.1002/nla.799.

[4] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, Adv. Des. Control 6, SIAM, Philadelphia, 2005, https://doi.org/10.1137/1.9780898718713.

[5] A. C. Antoulas, D. C. Sorensen, and Y. Zhou, *On the decay rate of Hankel singular values and related issues*, Systems Control Lett., 46 (2002), pp. 323–342, https://doi.org/10.1016/S0167-6911(02)00147-0.

[6] W. F. Arnold, III, and A. J. Laub, *Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations*, Proc. IEEE, 72 (1984), pp. 1746–1754.

[7] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Software Environ. Tools 11, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719581.

[8] J. Baker, M. Embree, and J. Sabino, *Fast singular value decay for Lyapunov solutions with nonnormal coefficients*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 656–668, https://doi.org/10.1137/140993867.

[9] H. T. Banks and K. Ito, *A numerical algorithm for optimal feedback gains in high dimensional linear quadratic regulator problems*, SIAM J. Control Optim., 29 (1991), pp. 499–515, https://doi.org/10.1137/0329029.

[10] E. Bänsch and P. Benner, *Stabilization of incompressible flow problems by Riccati-based feedback*, in Constrained Optimization and Optimal Control for Partial Differential Equations, G. Leugering, S. Engell, A. Griewank, M. Hinze, R. Rannacher, V. Schulz, M. Ulbrich, and S. Ulbrich, eds., Internat. Ser. Numer. Math. 160, Birkhäuser, Basel, 2012, pp. 5–20, https://doi.org/10.1007/978-3-0348-0133-1.

[11] E. Bänsch, P. Benner, J. Saak, and H. K. Weichelt, *Riccati-based boundary feedback stabilization of incompressible Navier–Stokes flows*, SIAM J. Sci. Comput., 37 (2015), pp. A832–A858, https://doi.org/10.1137/140980016.

[12] S. Barrachina, P. Benner, and E. S. Quintana-Ortí, *Efficient algorithms for generalized algebraic Bernoulli equations based on the matrix sign function*, Numer. Algorithms, 46 (2007), pp. 351–368, https://doi.org/10.1007/s11075-007-9143-x.

[13] P. Benner, *Computational methods for linear-quadratic optimization*, in Proceedings of the Workshop "Numerical Methods in Optimization" (Cortona, 1997). Rend. Circ. Mat. Palermo (2) Suppl. 58, Circ. Math. Palermo, Palermo, Italy, 1999, pp. 21–56.

[14] P. Benner, *Partial Stabilization of Descriptor Systems using Spectral Projectors*, in Numerical Linear Algebra in Signals, Systems and Control, P. Van Dooren, S. P. Bhattacharyya, R. H. Chan, V. Olshevsky, and A. Routray, eds., Lect. Notes Electr. Eng. 80, Springer, Dordrecht, The Netherlands, 2011, pp. 55–76, https://doi.org/10.1007/978-94-007-0602-6_3.

[15] P. Benner, M. Bollhöfer, D. Kressner, C. Mehl, and T. Stykel, *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory*, Springer, Cham, 2015, https://doi.org/10.1007/978-3-319-15260-8_1.

[16] P. Benner and Z. Bujanović, *On the solution of large-scale algebraic Riccati equations by using low-dimensional invariant subspaces*, Linear Algebra Appl., 488 (2016), pp. 430–459, https://doi.org/10.1016/j.laa.2015.09.027.

[17] P. Benner, Z. Bujanović, P. Kürschner, and J. Saak, *RADI: A low-rank ADI-type algorithm for large scale algebraic Riccati equations*, Numer. Math., 138 (2018), pp. 301–330, https://doi.org/10.1007/s00211-017-0907-5.

[18] P. Benner and R. Byers, *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Trans. Automat. Control, 43 (1998), pp. 101–107, https://doi.org/10.1109/9.654908.

[19] P. Benner and J. Heiland, *LQG-balanced truncation low-order controller for stabilization of laminar flows*, in Active Flow and Combustion Control 2014, R. King, ed., Notes Numer. Fluid Mech. Multidiscip. Des. 127, Springer, Cham, 2015, pp. 365–379, https://doi.org/10.1007/978-3-319-11967-0_22.

[20] P. Benner, M. Heinkenschloss, J. Saak, and H. K. Weichelt, *An inexact low-rank Newton-ADI method for large-scale algebraic Riccati equations*, Appl. Numer. Math., 108 (2016), pp. 125–142, https://doi.org/10.1016/j.apnum.2016.05.006.

[21] P. Benner, M. Heinkenschloss, J. Saak, and H. K. Weichelt, *Efficient solution of large-scale algebraic Riccati equations associated with index-2 DAEs via the inexact low-rank Newton-ADI method*, Appl. Numer. Math., 152 (2020), pp. 358–354, https://doi.org/10.1016/j.apnum.2019.11.016

[22] P. Benner, P. Kürschner, and J. Saak, *A reformulated low-rank ADI iteration with explicit residual factors*, Proc. Appl. Math. Mech., 13 (2013), pp. 585–586, https://doi.org/10.1002/pamm.201210308.

[23] P. Benner, P. Kürschner, and J. Saak, *Efficient handling of complex shift parameters in the low-rank Cholesky factor ADI method*, Numer. Algorithms, 62 (2013), pp. 225–251, https://doi.org/10.1007/s11075-012-9569-7.

[24] P. Benner, P. Kürschner, and J. Saak, *Self-generating and efficient shift parameters in ADI methods for large Lyapunov and Sylvester equations*, Electrn. Trans. Numer. Anal., 43 (2014), pp. 142–162.

[25] P. Benner, P. Kürschner, and J. Saak, *Frequency-limited balanced truncation with low-rank approximations*, SIAM J. Sci. Comput., 38 (2016), pp. A471–A499, https://doi.org/10.1137/15M1030911.

[26] P. Benner, P. Kürschner, and J. Saak, *Low-rank Newton-ADI methods for large nonsymmetric algebraic Riccati equations*, J. Frankl. Inst., 353 (2016), pp. 1147–1167, https://doi.org/10.1016/j.jfranklin.2015.04.016.

[27] P. Benner, J.-R. Li, and T. Penzl, *Numerical solution of large Lyapunov equations, Riccati equations, and linear-quadratic control problems*, Numer. Linear Algebra Appl., 15 (2008), pp. 755–777, https://doi.org/10.1002/nla.622.

[28] P. Benner, V. Mehrmann, and D. C. Sorensen, *Dimension Reduction of Large-Scale Systems*, Lect. Notes Comput. Sci. Eng. 45, Springer-Verlag, Berlin, Heidelberg, Germany, 2005.

[29] P. Benner and H. Mena, *Numerical solution of the infinite-dimensional LQR-problem and the associated differential Riccati equations*, J. Numer. Math., 26 (2018), pp. 1–20, https://doi.org/10.1515/jnma-2016-1039.

[30] P. Benner and J. Saak, *Efficient solution of large scale Lyapunov and Riccati equations arising in model order reduction problems*, Proc. Appl. Math. Mech., 8 (2008), pp. 10085–10088, https://doi.org/10.1002/pamm.200810085.

[31] P. Benner and J. Saak, *A Galerkin-Newton-ADI Method for Solving Large-Scale Algebraic Riccati Equations*, preprint SPP1253-090, SPP1253, 2010, http://www.am.uni-erlangen.de/home/spp1253/wiki/index.php/Preprints.

[32] P. Benner and J. Saak, *Numerical solution of large and sparse continuous time algebraic matrix Riccati and Lyapunov equations: A state of the art survey*, GAMM Mitt., 36 (2013), pp. 32–52, https://doi.org/10.1002/gamm.201310003.

[33] P. Benner and T. Stykel, *Numerical solution of projected algebraic Riccati equations*, SIAM J. Numer. Anal, 52 (2014), pp. 581–600, https://doi.org/10.1137/130923993.

[34] M. Berljafa and S. Güttel, *Generalized rational Krylov decompositions with an application to rational approximation*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 894–916, https://doi.org/10.1137/140998081.

[35] D. A. Bini, B. Iannazzo, and B. Meini, *Numerical Solution of Algebraic Riccati Equations*, Fundamentals of Algorithms 9, SIAM, Philadelphia, 2011, https://doi.org/10.1137/1.9781611972092.

[36] T. Bonin, H. Fassbender, A. Soppa, and M. Zaeh, *A fully adaptive rational global Arnoldi method for the model-order reduction of second-order MIMO systems with proportional damping*, Math. Comput. Simulat., 122 (2015), pp. 1–19, https://doi.org/10.1016/j.matcom.2015.08.017.

[37] J. L. Casti, *Dynamical Systems and Their Applications: Linear Theory*, Mathematics in Science and Engineering, Academic Press, New York, 1977.

[38] T. Çimen, *State-dependent Riccati equation (SDRE) control: A survey*, IFAC Proc. Vol., 41 (2008), pp. 3761—3775, https://doi.org/10.3182/20080706-5-KR-1001.00635.

[39] C. Choi and A. J. Laub, *Efficient matrix-valued algorithms for solving stiff Riccati differential equations*, IEEE Trans. Automat. Control, 35 (1990), pp. 770–776, https://doi.org/10.1109/9.57015.

[40] D. Chu, X. Liu, and V. Mehrmann, *A numerical method for computing the Hamiltonian Schur form*, Numer. Math., 105 (2006), pp. 375–412, https://doi.org/10.1007/s00211-006-0043-0.

[41] E. K. W. Chu, H. Y. Fan, and W. W. Lin, *A structure-preserving doubling algorithm for continuous-time algebraic Riccati equations*, Linear Algebra Appl., 396 (2005), pp. 55–80, https://doi.org/10.1016/j.laa.2004.10.010.

[42] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), 1, https://doi.org/10.1145/2049662.2049663.

[43] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408, https://doi.org/10.1137/0719025.

[44] U. B. Desai and D. Pal, *A transformation approach to stochastic model reduction*, IEEE Trans. Automat. Control, 29 (1984), pp. 1097–1100, https://doi.org/10.1109/TAC.1984.1103438.

[45] L. Dieci, *Numerical integration of the differential Riccati equation and some related issues*, SIAM J. Numer. Anal., 29 (1992), pp. 781–815, https://doi.org/10.1137/0729049.

[46] V. Druskin and L. A. Knizhnerman, *Extended Krylov subspaces: Approximation of the matrix square root and related functions*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 755–771, https://doi.org/10.1137/S0895479895292400.

[47] V. Druskin, L. A. Knizhnerman, and V. Simoncini, *Analysis of the rational Krylov subspace and ADI methods for solving the Lyapunov equation*, SIAM J. Numer. Anal., 49 (2011), pp. 1875–1898, https://doi.org/10.1137/100813257.

[48] V. Druskin and V. Simoncini, *Adaptive rational Krylov subspaces for large-scale dynamical systems*, Systems Cont. Lett., 60 (2011), pp. 546–560, https://doi.org/10.1016/j.sysconle.2011.04.013.

[49] V. Druskin, V. Simoncini, and M. Zaslavsky, *Adaptive tangential interpolation in rational Krylov subspaces for MIMO dynamical systems*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 476–498, https://doi.org/10.1137/120898784.

[50] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32, https://doi.org/10.1137/0917003.

[51] H. C. Elman and M. W. Rostami, *Efficient iterative algorithms for linear stability analysis of incompressible flows*, IMA J. Numer. Anal., 36 (2016), pp. 296–316, https://doi.org/10.1093/imanum.

[52] F. Feitzinger, T. Hylla, and E. W. Sachs, *Inexact Kleinman–Newton method for Riccati equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 272–288, https://doi.org/10.1137/070700978.

[53] F. D. Freitas and A. S. Costa, *Computationally efficient optimal control methods applied to power systems*, in Proceedings of the 20th International Conference on Power Industry Computer Applications, IEEE, Washington, DC, 1997, pp. 287–294, https://doi.org/10.1109/PICA.1997.599416.

[54] A. Frommer, K. Lund, and D. B. Szyld, *Block Krylov subspace methods for computing functions of matrices applied to multiple vectors*, Electron. Trans. Numer. Anal., 47 (2017), pp. 100–126.

[55] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed, Johns Hopkins University Press, Baltimore, MD, 2013.

[56] L. Grasedyck, *Existence of a low rank or H-matrix approximant to the solution of a Sylvester equation*, Numer. Linear Algebra Appl., 11 (2004), pp. 371–389, https://doi.org/10.1002/nla.366.

[57] S. Güttel, *Rational Krylov approximation of matrix functions: Numerical methods and optimal pole selection*, GAMM-Mitt., 36 (2013), pp. 8–31, https://doi.org/10.1002/gamm.201310002.

[58] C. He and V. Mehrmann, *Stabilization of large linear systems*, in Preprints of the European IEEE Workshop CMP'94 9Prague, 1994), L. Kulhavá, M. Kárný, and K. Warwick, eds., IEEE, Washington, DC, 1994, pp. 91–100.

[59] S. Hein, *MPC-LQG-Based Optimal Control of Parabolic PDEs*, Ph.D. thesis, TU Chemnitz, Chemnitz, Germany, 2009, http://archiv.tu-chemnitz.de/pub/2010/0013.

[60] M. Heinkenschloss, D. C. Sorensen, and K. Sun, *Balanced truncation model reduction for a class of descriptor systems with application to the Oseen equations*, SIAM J. Sci. Comput., 30 (2008), pp. 1038–1063, https://doi.org/10.1137/070681910.

[61] M. Heyouni and K. Jbilou, *An extended block Arnoldi algorithm for large-scale solutions of the continuous-time algebraic Riccati equation*, Electron. Trans. Numer. Anal., 33 (2009), pp. 53–62, http://etna.mcs.kent.edu/volumes/2001-2010/vol33/abstract.php?vol=33&pages=53-62.

[62] T. Hylla, *Extension of Inexact Kleinman-Newton Methods to a General Monotonicity Preserving Convergence Theory*, Dissertation, Universität Trier, Trier, Germany, 2011, http://ubt.opus.hbz-nrw.de/volltexte/2011/643.

[63] I. M. Jaimoukha and E. M. Kasenally, *Krylov subspace methods for solving large Lyapunov equations*, SIAM J. Numer. Anal., 31 (1994), pp. 227–251, https://doi.org/10.1137/0731012.

[64] K. Jbilou, *An Arnoldi based algorithm for large algebraic Riccati equations*, Appl. Math. Lett., 19 (2006), pp. 437–444, https://doi.org/10.1016/j.aml.2005.07.001.

[65] K. Jbilou, A. Messaoudi, and H. Sadok, *Global FOM and GMRES algorithms for matrix equations*, Appl. Numer. Math., 31 (1999), pp. 49–63, https://doi.org/10.1016/S0168-9274(98)00094-4.

[66] E. A. Jonckheere and L. M. Silverman, *A new set of invariants for linear systems—application to reduced order compensator design*, IEEE Trans. Automat. Control, 28 (1983), pp. 953–964, https://doi.org/10.1109/TAC.1983.1103159.

[67] T. Kailath, *Some Chandrasekhar-type algorithms for quadratic regulators*, in Proceedings of the 1972 IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes, IEEE, Washington, DC, 1972, pp. 219–223, https://doi.org/10.1109/CDC.1972.268990.

[68] D. Kleinman, *On an iterative technique for Riccati equation computations*, IEEE Trans. Automat. Control, 13 (1968), pp. 114–115, https://doi.org/10.1109/TAC.1968.1098829.

[69] L. Knizhnerman and V. Simoncini, *Convergence analysis of the extended Krylov subspace method for the Lyapunov equation*, Numer. Math., 118 (2011), pp. 567–586, https://doi.org/10.1007/s00211-011-0366-3.

[70] L. A. Knizhnerman and V. Simoncini, *A new investigation of the extended Krylov subspace method for matrix function evaluations*, Numer. Linear Algebra Appl., 17 (2010), pp. 615–638, https://doi.org/10.1002/nla.652.

[71] D. Kressner, P. Kürschner, and S. Massei, *Low-rank updates and divide-and-conquer methods for quadratic matrix equations*, Numer. Alg., (2019), pp. 1–25, https://doi.org/10.1007/s11075-019-00776-w.

[72] P. Kürschner, *Efficient Low-Rank Solution of Large-Scale Matrix Equations*, Ph.D. thesis, Otto von Guericke Universität, Magdeburg, Germany, 2016, http://hdl.handle.net/11858/00-001M-0000-0029-CE18-2.

[73] P. Kürschner and M. Freitag, *Inexact Methods for the Low Rank Solution to Large Scale Lyapunov Equations*, preprint, https://arxiv.org/abs/1809.06903, 2018.

[74] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*, Oxford Science Publications, The Clarendon Press, Oxford University Press, New York, 1995.

[75] N. Lang, H. Mena, and J. Saak, *On the benefits of the LDL^T factorization for large-scale differential matrix equation solvers*, Linear Algebra Appl., 480 (2015), pp. 44–71, https://doi.org/10.1016/j.laa.2015.04.006.

[76] A. Lanzon, Y. Feng, and B. D. O. Anderson, *An iterative algorithm to solve Algebraic Riccati Equations with an indefinite quadratic term*, in 2007 European Control Conference (ECC), 2007, pp. 3033–3039.

[77] A. J. Laub, *A Schur Method for Solving Algebraic Riccati Equations*, IEEE Trans. Automat. Control, 24 (1979), pp. 913–921, https://doi.org/10.1109/TAC.1979.1102178.

[78] J.-R. Li, *Model reduction of large linear systems via low rank system Gramians*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[79] J.-R. Li and J. White, *Low rank solution of Lyapunov equations*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 260–280, https://doi.org/10.1137/S0895479801384937.

[80] T. Li, E. K. Chu, W. W. Lin, and P. C. Y. Weng, *Solving large-scale continuous-time algebraic Riccati equations by doubling*, J. Comput. Appl. Math., 237 (2013), pp. 373–383, https://doi.org/10.1016/j.cam.2012.06.006.

[81] Y. Lin and V. Simoncini, *A new subspace iteration method for the algebraic Riccati equation*, Numer. Linear Algebra Appl., 22 (2015), pp. 26–47, https://doi.org/10.1002/nla.1936.

[82] A. Locatelli, *Optimal Control: An Introduction*, Birkhäuser, Basel, 2001.

[83] A. Massoudi, M. R. Opmeer, and T. Reis, *Analysis of an iteration method for the algebraic Riccati equation*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 624–648, https://doi.org/10.1137/140985792.

[84] V. Mehrmann and E. Tan, *Defect Correction Methods for the Solution of Algebraic Riccati Equations*, IEEE Trans. Automat. Control, 33 (1988), pp. 695–698, https://doi.org/10.1109/9.1282.

[85] C. Moosmann, E. B. Rudnyi, A. Greiner, and J. G. Korvink, *Model order reduction for linear convective thermal flow*, in THERMINIC 2004, Sophia Antipolis, France, 2004, pp. 317–321.

[86] P. C. Opdenacker and E. A. Jonckheere, *A contraction mapping preserving balanced reduction scheme and its infinity norm error bounds*, IEEE Trans. Circuits Syst., 35 (1988), pp. 184–189, https://doi.org/10.1109/31.1720.

[87] M. R. Opmeer, *Decay of Hankel singular values of analytic control systems*, Systems Control Lett., 59 (2010), pp. 635–638, https://doi.org/10.1016/j.sysconle.2010.07.009.

[88] M. R. Opmeer, *Decay of singular values of the Gramians of infinite-dimensional systems*, in 2015 European Control Conference (ECC), 2015, pp. 1183–1188, https://doi.org/10.1109/ECC.2015.7330700.

[89] T. Penzl, *A cyclic low-rank Smith method for large sparse Lyapunov equations*, SIAM J. Sci. Comput., 21 (2000), pp. 1401–1418, https://doi.org/10.1137/S1064827598347666.

[90] A. Ruhe, *Rational Krylov sequence methods for eigenvalue computation*, Linear Algebra Appl., 58 (1984), pp. 391–405, https://doi.org/10.1016/0024-3795(84)90221-0.

[91] A. Ruhe, *The Rational Krylov algorithm for nonsymmetric Eigenvalue problems.* III*: Complex shifts for real matrices*, BIT, 34 (1994), pp. 165–176, https://doi.org/10.1007/BF01935024.

[92] D. L. Russell, *Mathematics of Finite-Dimensional Control Systems*, Lect. Notes Pure Appl. Math. 43, Marcel Dekker, New York, 1979.

[93] Y. Saad, *Numerical solution of large Lyapunov equation*, in Signal Processing, Scattering, Operator Theory and Numerical Methods, M. A. Kaashoek, J. H. van Schuppen, and A. C. M. Ran, eds., Progr. Systems Control Theory 5, Birkhäuser, Basel, 1990, pp. 503–511.

[94] J. Saak, *Efficient Numerical Solution of Large Scale Algebraic Matrix Equations in PDE Control and Model Order Reduction*, Ph.D. thesis, TU Chemnitz, Chemnitz, Germany, 2009, http://nbn-resolving.de/urn:nbn:de:bsz:ch1-200901642.

[95] J. Saak, M. Köhler, and P. Benner, *M-M.E.S.S.*-1.0.1—*The Matrix Equations Sparse Solvers Library*, 2016, https://doi.org/10.5281/zenodo.50575; see also https://www.mpi-magdeburg.mpg.de/projects/mess.

[96] J. Saak, P. Kürscher, Z. Bujanović, and P. Benner, *Collected MATLAB solvers for large-scale AREs*, v. 1.0, Zenodo, https://doi.org/10.5281/zenodo.3662519.

[97] J. Sabino, *Solution of Large-Scale Lyapunov Equations via the Block Modified Smith Method*, Ph.D. thesis, Rice University, Houston, TX, 2007, http://www.caam.rice.edu/tech_reports/2006/TR06-08.pdf.

[98] M. Schmidt, *Systematic Discretization of Input/Output Maps and Other Contributions to the Control of Distributed Parameter Systems*, Ph.D. thesis, Technische Universität Berlin, Berlin, Germany, 2007.

[99] V. Sima, *Algorithms for Linear-Quadratic Optimization*, Pure Appl. Math. 200, Marcel Dekker, New York, 1996.

[100] V. Simoncini, *A new iterative method for solving large-scale Lyapunov matrix equations*, SIAM J. Sci. Comput., 29 (2007), pp. 1268–1288, https://doi.org/10.1137/06066120X.

[101] V. Simoncini, *Analysis of the rational Krylov subspace projection method for large-scale algebraic Riccati equations*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1655–1674, https://doi.org/10.1137/16M1059382.

[102] V. Simoncini, *Computational methods for linear matrix equations*, SIAM Rev., 58 (2016), pp. 377–441, https://doi.org/10.1137/130912839.

[103] V. Simoncini, D. Szyld, and M. Monsalve, *On two numerical methods for the solution of large-scale algebraic Riccati equations*, IMA J. Numer. Anal., 34 (2014), pp. 904–920, https://doi.org/10.1093/imanum.

[104] T. Stykel and V. Simoncini, *Krylov subspace methods for projected Lyapunov equations*, Appl. Numer. Math., 62 (2012), pp. 35–50, https://doi.org/10.1016/j.apnum.2011.09.007.

[105] K. Sun, *Model Order Reduction and Domain Decomposition for Large-Scale Dynamical Systems*, Ph.D. thesis, Rice University, Houston, TX, 2008, http://search.proquest.com/docview/304507831.

[106] The MORwiki Community, *MORwiki: Model Order Reduction Wiki*, http://modelreduction.org.

[107] A. Varga, *On stabilization methods of descriptor systems*, Systems Control Lett., 24 (1995), pp. 133–138, https://doi.org/10.1016/0167-6911(94)00017-P.

[108] E. L. Wachspress, *The ADI Model Problem*, Springer, New York, 2013, https://doi.org/10.1007/978-1-4614-5122-8.

[109] H. Weichelt, *Numerical Aspects of Flow Stabilization by Riccati Feedback*, Ph.D. thesis, Otto von Guericke Universität, Magdeburg, Germany, 2016, http://nbn-resolving.de/urn:nbn:de:gbv:ma9:1-8693.

[110] T. Wolf, $\mathcal{H}_2$ *Pseudo-Optimal Model Order Reduction*, Ph.D. thesis, Technische Universität München, München, Germany, 2015, https://d-nb.info/1064075568/34.

[111] N. Wong and V. Balakrishnan, *Fast positive-real balanced truncation via quadratic alternating direction implicit iteration*, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., 26 (2007), pp. 1725–1731, https://doi.org/10.1109/TCAD.2007.895617.