Solving Algebraic Riccati Equations on Parallel Computers Using Newton's Method with Exact Line Search

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

Report 98–05

Berichte aus der Technomathematik

Report 98–05 August 1998

Solving Algebraic Riccati Equations on Parallel Computers Using Newton's Method with Exact Line Search

Peter Benner,[a,2] Ralph Byers,[b,3] Enrique S. Quintana-Ortí,[c,1,2] Gregorio Quintana-Ortí[c,1]

[a]*Zentrum für Technomathematik, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D–28334 Bremen, Germany; e-mail: benner@math.uni-bremen.de*

[b]*Department of Mathematics, University of Kansas, Lawrence, KS 66045, USA; e-mail: byers@math.ukans.edu*

[c]*Departamento de Informática, Universidad Jaime I, 12.071–Castellón, Spain; e-mails: quintana@inf.uji.es, gquintan@inf.uji.es*

Abstract

We investigate the numerical solution of continuous-time algebraic Riccati equations via Newton's method on serial and parallel computers with distributed memory. We apply and extend the available theory for Newton's method endowed with exact line search to accelerate convergence. We also discuss a new stopping criterion based on recent observations regarding condition and error estimates. In each iteration step of Newton's method a stable Lyapunov equation has too be solved. We propose to solve these Lyapunov equations using iterative schemes for computing the matrix sign function. This approach can be efficiently implemented on parallel computers using ScaLAPACK. Numerical experiments on an IBM SP2 multicomputer report the accuracy, scalability, and speed-up of the implemented algorithms.

Key words: continuous-time algebraic Riccati equation, Lyapunov equation, Newton's method, matrix sign function, line search.

---

[1] Partially supported by the CICYT project No. TIC96-1062-C03-C03.
[2] Partially supported by the DAAD *Acciones Integradas Hispano–Alemanas* program.
[3] Partially supported by National Science Foundation awards DMS-9628626, CCR-9732671 and by the NSF EPSCoR/K*STAR program through the Center for Advanced Scientific Computing.

28 August 1998

1 Introduction

Consider the *(generalized) continuous-time algebraic Riccati equation* (CARE)

$$0 = \mathcal{R}(X) := Q + A^T XE + E^T XA - E^T XGXE, \tag{1}$$

where $A, E, G, Q, X \in \mathbb{R}^{n\times n}$, $Q = Q^T$, $G = G^T$, and $X = X^T$ is the sought-after solution. Throughout this paper we assume that E is nonsingular. In principle, by inverting, (1) can be reduced to a standard CARE, i.e., $E = I_n$, where I_n denotes the $n \times n$ identity matrix. This is avoided here in order not to introduce unnecessary rounding errors caused by a possible ill-conditioning of E.

Algebraic Riccati equations are of fundamental importance in many analysis and synthesis algorithms in control theory. They arise naturally in optimal and robust control problems driven by linear autonomous first-order ordinary differential equations (ODE) such as the linear-quadratic regulator or the $\mathcal{H}_2$-/$\mathcal{H}_\infty$-control problems; see, e.g., [1,39,44,47,54,59]. As many methods of nonlinear control use the linear system obtained from a linearization of the nonlinear ODE around a working point, these methods also require the sound and efficient solution of equations of the form (1). The generalized equations of type (1) with $E \neq I_n$ arise naturally from control systems driven by second-order ODEs, descriptor systems, or partial differential equations (PDE). See, e.g., [20,41,43,44,50,54]. In particular, in recent years, model reduction for large-scale control problems has become one of the most important issues in systems and control theory. Most of the recent algorithms in this area rely in one way or another on the solution of equations of the form (1); see, e.g., [51,59] and the overview given in [52]. Moreover, factorization techniques for rational matrix functions may lead to dense, unstructured, and large equations of the form (1), see, e.g., [57].

In most applications from control theory, the desired solution of the CARE (hereafter X_*) is *stabilizing* in the sense that all the eigenvalues of the matrix pencil $E - \lambda(A - GX_*E)$ have negative real part. This will be denoted by $\sigma(E, A - GX_*E) \subset \mathbb{C}^-$. Throughout this paper we will assume that such a stabilizing solution exists. Note that if it exists, it is unique [39]. For instance, if $(E^{-1}A, E^{-1}GE^{-T})$ is stabilizable, $(E^{-1}A, Q)$ is detectable, and G, Q are positive semidefinite (denoted by $G \geq 0$, $Q \geq 0$), then the stabilizing solution X_* exists, is unique, and, in addition, $X_* \geq 0$ [39]. But note that these are only sufficient conditions and by no means necessary. In particular, X_* may be indefinite if any of the above conditions is dropped and the stabilizing solution still exists.

The need for parallel computing in this area can be seen from the fact that already for control systems with state-space dimension $n = 1000$, (1) represents a set of nonlinear equations with 500500 unknowns (having already exploited the symmetry of X_*). Systems of such a dimension driven by ODEs are not uncommon in chemical engineering applications, are standard for second order systems, and represent rather coarse grids when derived from the discretization of a PDE; see, e.g., [19,22,23,43,45,50].

One of the oldest and most well-known numerical methods for solving the CARE is Newton's method (or Kleinman's iteration) [38]. This method iteratively approximates the solution of the CARE and requires, at each iteration, the solution of a Lyapunov equation of the form

$$0 = \tilde{Q} + \tilde{A}^T \tilde{Y} \tilde{E} + \tilde{E}^T \tilde{Y} \tilde{A}, \tag{2}$$

where $\tilde{A}, \tilde{E}, \tilde{Q}, \tilde{Y} \in \mathbb{R}^{n\times n}$, $\sigma(\tilde{E}, \tilde{A}) \subset \mathbb{C}^-$, and $\tilde{Q} = \tilde{Q}^T$.

Numerical methods for the standard Lyapunov equation ($\tilde{E} = I_n$) are studied in [7,30]. In the initial stage of these methods, $\tilde{A}$ is reduced to real Schur form by means of the QR algorithm [27]. This is followed by a quite less expensive back substitution process. The difficulties of parallelizing the QR algorithm on distributed memory architectures are twofold: first, the algorithm is composed of fine-grain computations with a low ratio of computation/communication; second, the use of traditional block scattered data layouts leads to an unbalanced distribution of the computational load [31,32]. Different approaches have been proposed to avoid these drawbacks, namely, multishift techniques which allow for larger grain computations [32,58], and a block Hankel distribution of the matrix that improves the balancing of the computational load [31]. The experimental studies, however, report parallelism and scalability results which are not as good as those of the kernels arising in linear systems, e.g., matrix factorizations, triangular linear systems solvers, etc. See, e.g., [14,18].

In the generalized case, each iteration of Newton's method requires the solution of a generalized Lyapunov equation ($\tilde{E} \neq I_n$). Numerical methods for this equation are proposed in [24,46]. There, the matrix pair $(\tilde{E}, \tilde{A})$ is first reduced to generalized real Schur form by means of the QZ algorithm [27]. This stage is followed by a back substitution procedure. Unfortunately, we are not aware of any available implementation of the QZ algorithm for parallel computers with distributed memory. Moreover, since both the QR and the QZ algorithms are composed of similar computations, the same problems as observed for the QR algorithm can be expected in the parallelization of the QZ algorithm.

In this paper we study Newton's method for solving the CARE. We will use an exact line search technique to accelerate convergence in early stages of the iteration. Our Lyapunov solvers are based on matrix sign function computations and only require computational kernels like matrix products, linear system solvers, and matrix inversions. Very efficient implementations of these operations for parallel computers with distributed architectures exist within ScaLAPACK [14]. Thus, we avoid the parallelization difficulties in the QR/QZ algorithms.

In Section 2 we summarize Newton's method with exact line search for solving the CARE and discuss the convergence theory. Moreover, we review the condition number of the CARE for $E = I_n$ and present an approximate condition number of the CARE for the case $E \neq I_n$. Based on these condition numbers and a recently given a posteriori residual bound for the relative error of an approximate solution of (1) we propose a reliable stopping criterion for Newton's method in order to solve the CARE to highest possible accuracy. From these considerations we also obtain a forward error bound for the relative error of the computed solution. In Section 3 we describe our serial Lyapunov solvers based on matrix sign function iterations. A brief study of the computational and communication cost of the parallel implementations is presented in Section 4. In Section 5 we analyze the experimental accuracy, performance, and scalability of our solvers on a parallel distributed architecture, the IBM SP2. Concluding remarks are given in Section 6.

2 Newton's Method for Continuous-Time Algebraic Riccati Equations

2.1 Newton's method and exact line search

In [38], Kleinman shows that Newton's method applied to the standard CARE, converges to the desired stabilizing solution of the CARE. (See Theorem 1 below.) Variants of Newton's method for solving the generalized CARE are presented in [4,9,10,44]:

Algorithm 1 (Newton's method for the generalized CARE).

```
Input: $A, E, G, Q \in \mathbb{R}^{n\times n}$, $G = G^T$, $Q = Q^T$, $X_0 = X_0^T$ – an initial guess.
Output: Approximate solution $X_{j+1} \in \mathbb{R}^{n\times n}$ of (1).
FOR $j = 0, 1, 2, \ldots$ "until convergence"
  (1) $A_j = A - GX_jE$.
  (2) Solve for $N_j$ the generalized Lyapunov equation
        $0 = \mathcal{R}(X_j) + A_j^T N_j E + E^T N_j A_j.$
  (3) $X_{j+1} = X_j + N_j$.
END FOR
```

We will discuss possible stopping criteria for the above algorithm in Section 2.4. The properties of Algorithm 1 are summarized in the following theorem [38,44,39].

Theorem 1 *If E is nonsingular, $G \geq 0$, $(E^{-1}A, E^{-1}GE^{-T})$ is stabilizable, the unique stabilizing solution X_* exists, and X_0 is stabilizing, then for the iterates produced by Algorithm 1 we have:*

(1) All iterates X_j are stabilizing, i.e., $\sigma(E, A-GX_jE) \subset \mathbb{C}^-$ for all $j \in \mathbb{N}_0$.
(2) $X_* \leq \ldots \leq X_{j+1} \leq X_j \leq \ldots \leq X_1$.
(3) $\lim_{j\to\infty} X_j = X_*$.

(4) There exists a constant $\gamma > 0$ *such that*

$$\|X_{j+1} - X_*\| \leq \gamma \|X_j - X_*\|^2, \qquad j \geq 1,$$

i.e., the X_j *converge globally and quadratic to* X_*.

Remark 2 *A proof for the results collected in Theorem 1 for the case* $E = I_n$*, using the same assumptions as above, can be found in [39]. The above theorem for* $E \neq I_n$ *nonsingular is a trivial corollary of the results in [39] using the equivalence of (1) to*

$$0 = Q + \tilde{A}^T \tilde{X} + \tilde{X}\tilde{A} - \tilde{X}\tilde{G}\tilde{X}$$

obtained by setting $\tilde{A} := E^{-1}A$*,* $\tilde{G} := E^{-1}GE^{-T}$*, and* $\tilde{X} := E^T X E$ *[9]. This equivalence is also used by Mehrmann in [44] to prove a version of Theorem 1 using the additional assumption* $Q > 0$*, i.e., assuming positive definiteness of* Q.

Note that part (ii) (monotone convergence) of Theorem 1 in general only holds for $j \geq 1$. Nevertheless, the result holds for $j = 0$ if X_0 is close enough to X_*; see [9, Section 4.1.2] and [37]. In general it is not unusual to have $X_0 < X_1$ as a simple scalar example shows: let

$$A = 0, \quad E = 1, \quad G = 1, \quad Q = 10^{-2}, \quad X_0 = 10^{-4}.$$

Then $X_* = 0.1$ and $X_1 \approx 50 \gg X_0$. This shows one of the major difficulties of Newton's method: sometimes the first step is an enormous leap away from X_0 and X_* and only returns slowly afterwards. Figure 1 demonstrates this by displaying the relative errors $\|X_* - X_j\|_2 / \|X_*\|_2$ during the iterations of Newton's method applied to the above example.

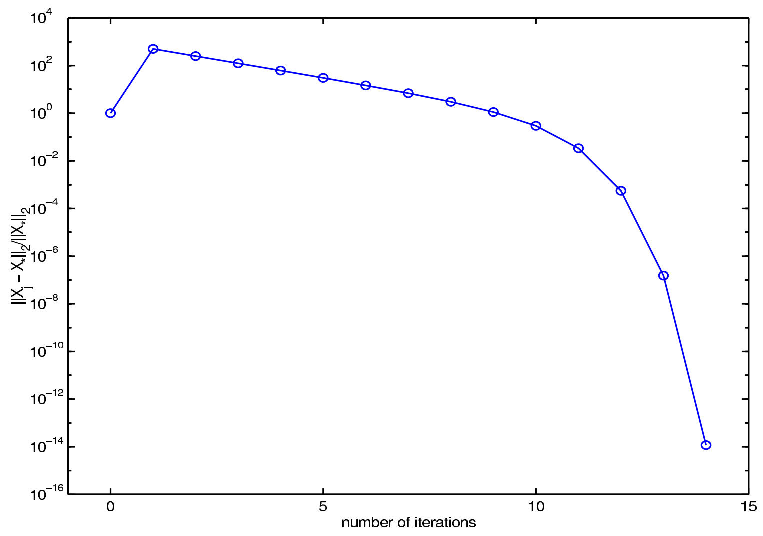


Fig. 1. Slow initial convergence of Newton's method.

The problem with the first step can be overcome and convergence of Newton's method can be accelerated by means of an exact line search [9,10]. Specifically, Step 3. of Algorithm 1 is modified to

$$X_{j+1} = X_j + t_j N_j, \tag{3}$$

where $t_j \in [0,2]$ is a local minimizer of

$$\begin{aligned} f_j(t) &= \|\mathcal{R}(X_j)\|_F^2 = \operatorname{trace}\left(\mathcal{R}(X_j + tN_j)^2\right) \\ &= \operatorname{trace}\left(\mathcal{R}(X_j)^2\right)(1-t)^2 - 2\operatorname{trace}\left(\mathcal{R}(X_j)V_j\right)(1-t)t^2 \\ &\quad + \operatorname{trace}\left(V_j^2\right)t^4, \end{aligned} \tag{4}$$

and $V_j = E^T N_j G N_j E$. In [9,10] it is proved that such a minimizer exists in $[0,2]$ and minimizes the Frobenius norm of the next residual, denoted by $\|\mathcal{R}(X_j + tN_j)\|_F$. The restriction to the interval $[0,2]$ is necessary in order to ensure that the X_j remain stabilizing; see [9,8,10]. Although the computation of the line search increases the cost of Newton's method (mainly because of the computation of V_j), in practice this overhead is largely compensated by a reduction in the number of iterations of Newton's method and therefore in the overall cost of the solver.

Also note that in most control applications, G is given in factored form $G = BB^T$ with $B \in \mathbb{R}^{n\times m}$. If this is exploited in the computation of V_j and $m \ll n$, then the additional cost caused by the line search procedure is negligible; for details see the next section.

The following theorem taken from [9] summarizes the convergence properties of the resulting algorithm as derived in [9,10].

Theorem 3 *Under the assumptions of Theorem 1, assuming in addition that* $(E^{-1}A, E^{-1}GE^{-T})$ *is controllable and that the local minimizer* $t_j \in [0,2]$ *of* f_j *in (4) satisfies* $t_j \geq t_L > 0$ *for all* $j \in \mathbb{N}_0$, *where* $t_L > 0$ *is a constant tolerance threshold, then the sequence of approximate solutions* X_j *produced by Algorithm 1 with Step 3. replaced by (3) has the following properties:*

(i) X_j *is stabilizing for all* $j \in \mathbb{N}_0$.

(ii) $\|\mathcal{R}(X_{j+1})\|_F \leq \|\mathcal{R}(X_j)\|_F$ *and equality holds if and only if* $\mathcal{R}(X_j) = 0$.

(iii) $\lim_{j\to\infty} \mathcal{R}(X_j) = 0$.

(iv) $\lim_{j\to\infty} X_j = X_*$.

(v) In a neighbourhood of X_*, *convergence is quadratic.*

(vi) $\lim_{j\to\infty} t_j = 1$.

The assumptions of the above theorem are stronger than those of Theorem 1 as controllability is required. It is conjectured but not proved in [9,10] that this can be relaxed to stabilizability. The controllability assumption is needed in [10] to show that Property (iii) in the above theorem holds. Further note that (i) can be proved using the assumptions of Theorem 1 without the additional assumptions used above.

Global convergence of the residuals to zero without assuming controllability may be ensured following a strategy based on the *Armijo rule* [3]. Here, we will follow the presentation used in [17,33]. Let $f : \mathbb{R}^\ell \to \mathbb{R}$ be twice continuously differentiable. If $x_j \in \mathbb{R}^\ell$ is an approximation to a local minimizer of $f(x)$ and p_j is a search direction for f from x_j, then a step size parameter t_j is accepted if it satisfies the sufficient decrease criterion given by the Armijo rule

$$f(x_j + tp_j) \leq f(x_j) + \alpha t_j (\nabla f(x_j))^T p_j, \tag{5}$$

where $\alpha \in (0,1)$. In order to translate (5) to the situation here, we use the usual embedding of $\mathbb{R}^{n\times n}$ into $\mathbb{R}^{n^2}$ given by the map vec : $\mathbb{R}^{n\times n} \longrightarrow \mathbb{R}^{n^2}$,

$$\mathrm{vec}\,(X) := [x_{11}, x_{21}, \ldots, x_{1n}, x_{12}, \ldots, x_{n2}, \ldots, x_{1n}, \ldots, x_{nn}]$$

which is obtained by consecutively stacking the columns of a matrix into a vector (see, e.g., [40, Chapter 12]). Setting $x := \mathrm{vec}\,(X)$ and

$$\begin{aligned} f(x) &= \frac{1}{2}\mathrm{vec}\,(\mathcal{R}\,(X))^T\,\mathrm{vec}\,(\mathcal{R}\,(X)) \\ &= \frac{1}{2}\mathrm{trace}\left(\mathcal{R}\,(X)^T\,\mathcal{R}\,(X)\right) = \frac{1}{2}\|\mathcal{R}\,(X)\,\|_F^2, \end{aligned}$$

it can be shown (see, e.g., [26]) that

$$\nabla f(\mathrm{vec}\,(X_j)) = \Gamma_j^T \mathrm{vec}\,(\mathcal{R}\,(X_j))\,.$$

Here, $\Gamma_j = (A - GX_jE)^T \otimes E^T + E^T \otimes (A - GX_jE)^T$ denotes the matrix representation of the Lyapunov operator, mapping a symmetric matrix $Z \in \mathbb{R}^{n\times n}$ to $(A - GX_jE)^T ZE + E^T Z(A - GX_jE)$. As all X_j are stabilizing, it follows that all Γ_j are nonsingular (see, e.g., [40, Chapter 13]). Hence,

$$p_j := \mathrm{vec}\,(N_j) = -\Gamma_j^{-1}\mathrm{vec}\,(\mathcal{R}\,(X_j))$$

is well defined and we obtain

$$\begin{aligned} (\nabla f(x_j))^T p_j &= -\mathrm{vec}\,(\mathcal{R}\,(X_j))^T\,\Gamma_j\Gamma_j^{-1}\mathrm{vec}\,(\mathcal{R}\,(X_j)) \\ &= -\mathrm{vec}\,(\mathcal{R}\,(X_j))^T\,\mathrm{vec}\,(\mathcal{R}\,(X_j)) = -\|\mathcal{R}\,(X_j)\,\|_F^2 < 0. \end{aligned} \tag{6}$$

From (6) it can be deduced [17] that the Newton direction provides a *descent direction* for $f(\mathrm{vec}\,(X))$ from X_j. Moreover, the Armijo rule (5) translates to

$$\|\mathcal{R}\,(X_j + t_jN_j)\,\|_F \leq \sqrt{1 - 2\alpha t_j}\|\mathcal{R}\,(X_j)\,\|_F. \tag{7}$$

If we choose a constant $\alpha \in (0, \frac{1}{4})$ and if the step sizes t_j are chosen from $[\,t_L, 2\,]$ for a given (small) constant $t_L > 0$, then

$$0 < 1 - 2\alpha t_j \leq 1 - 2\alpha t_L := \gamma^2 < 1 \tag{8}$$

for all $j = 0, 1, 2, \ldots$. Hence,

$$\|\mathcal{R}(X_j)\|_F \leq \gamma^j \|\mathcal{R}(X_0)\|_F. \tag{9}$$

As $\gamma < 1$, $\lim_{j\to\infty} \|\mathcal{R}(X_j)\|_F = 0$, i.e., we obtain global convergence of $\mathcal{R}(X_j)$ to zero. But note that this does not necessarily imply $X_j \to X_*$. It is currently unclear what are the necessary conditions to ensure this. In practice, however, convergence of $\mathcal{R}(X_j)$ always yields $X_j \to X_*$.
Note that (7) can be checked using scalar operations only by evaluating f_j using (4) without having to form the residual matrix $\mathcal{R}(X_j + t_j N_j)$.
Besides ensuring theoretical convergence, requiring (7) is also reasonable as otherwise, a stagnation of the line search method is possible. The same arguments can be used to explain why the assumption $t_j \geq t_L$ is not only an additional assumption but should be a requirement: choosing t_j too small will result in almost no progress towards the attractor of the iteration. How these two requirements can be realized in an actual implementation is described in the next section.
Remark 4 *Theorem 3 and the derivation of global convergence remain valid if A is stable and G is* negative *semidefinite; see [8]. This type of CAREs plays an important role in $\mathcal{H}_\infty$ control and model reduction; see, e.g., [59].*

2.2 Remarks and implementation details

In many control applications, the CARE (1) can be written in the form

$$0 = \mathcal{R}(X) := Q + A^T X E + E^T X A - X(B_1 B_1^T - B_2 B_2^T)X, \tag{10}$$

where $B_k \in \mathbb{R}^{n \times m_k}$, $k = 1, 2$. For instance, the standard CARE from the linear-quadratic regulator problem is in the form (10) by setting $B_2 = 0$ while the case $B_k \neq 0$ for $k = 1$ *and* $k = 2$ arises frequently in robust control problems; see, e.g., [59] and the references given therein. In case $m := m_1 + m_2 > n/2$, it is more efficient to compute $G = B_1 B_1^T - B_2 B_2^T$ in advance if no line searches are used [9]. On the other hand, the computation of V_j for the exact line search is always cheaper if the factors B_k are stored rather than G if $m \leq n$. In particular, if $m \ll n$, using the factors, the additional cost for forming V_j and hence for the exact line search is $\mathcal{O}(n^2)$ and therefore negligible; see [9] for details of the computations and also for a procedure for computing t_j, avoiding the explicit computation of V_j.
The rate of convergence of Newton's method strongly depends on the distance between the initial guess X_0 and X_*. A feasible initial guess can be obtained by procedures which are as expensive as one iteration of Newton's method [56]. Nevertheless, such solutions often lie far from X_* and a high number of iterations of Newton's method may be required to converge. A different approach consists of finding an initial solution by means of a CARE solver (the Schur vector method [42], the matrix sign function [49], etc.), but this

requires solving the CARE itself. Newton's method is used most frequently used for iterative refinement of approximate solutions computed by some other method. Using the exact line search proposed in the last section often brings the cost of Newton's method down to that of the Schur vector method or even below that; see [9,8] and Section 5. Also, if A is already stable, $X_0 = 0$ is a feasible initial guess and Newton's method with line search becomes a competitive alternative as a solver for CAREs; see [8].

The assumptions of Theorem 3 and (7) are ensured by employing the following considerations:

- Set a lower bound t_L for the step size parameters t_j. (Numerical experiments indicate that $t_L = 10^{-4}$ is a reasonable choice). Select the parameter $\alpha \in (0, \frac{1}{4})$ for the Armijo rule. In concurrent optimization literature [17,33], $\alpha = 10^{-4}$ is proposed. Together with t_L as above, γ^2 in (8) is $\approx 1 - 10^{-8}$ which may yield very slow convergence — something we try to avoid by using line searches. Therefore we propose here a much larger value for α. In our experiments, $\alpha = 0.2$ worked very well.
- In each step, after having computed the local minimizer $t_j \in [0, 2]$, we set $t_j = \max\{t_j, t_L\}$.
- The condition (7) can be checked by evaluating $f_j(t)$ using (4) rather than by forming the residual matrix explicitly. If (7) is not satisfied in some step, there are several options:
  - · Find another line search parameter t_j by a backtracking strategy as described in [17];
  - · decrease α or t_L;
  - · perform a standard Newton step with $t_j = 1$ — this may lead to a violation of Theorem 3.

  The second option usually results in a stagnation of the iteration as no significant progress will be made. The third option can be considered as restarting the Newton iteration from a new initial guess. Note that the convergence theory given in Theorems 1 and 3 ensures that the new "starting guess" is again stabilizing. One should limit the number of allowed restarts and run the Newton iteration without line search if this number is exceeded. This guarantees global convergence for the overall process.

In addition to the above, we also employ a criterion in order to overcome possible stagnation. This criterion can be written as follows:

$$\|\mathcal{R}(X_j + t_j N_j)\|_F < tol_S \|\mathcal{R}(X_{j-j_S})\|_F. \tag{11}$$

For this, we only need to keep the norms of the last j_S residuals. In case of stagnation we again "restart" the method using a Newton step. From numerical experience, we suggest to use $j_S = 2$, $tol_S = 0.9$.

Each line search requires computing a local minimizer of the quartic polynomial $f_j(t) = \|\mathcal{R}(X_j + tN_j)\|_F^2$ over the interval $[0, 2]$. The Newton step N_j corresponds to $t = 1$. It is typical for the minimizer to be near $t = 1$. Using (4), it can be shown that $f_j(t)$ is non-increasing at $t = 0$ and nondecreasing at $t = 2$ [8–10] and, in particular, all local minimizers—even endpoints—are roots of the derivative $f_j'(t)$. It also follows from (4) that if $t = 0$ is a local minimizer, then $\mathcal{R}(X_j) = 0$, $X_j = X_*$, $N_j = 0$, and $f_j(t) \equiv 0$ [8–10]. It is possible but unusual for $t = 2$ to be a local minimizer. However, there are circumstances in which one would expect a minimizer near $t = 2$; see the following remark.

Remark 5 *In [29] it is shown that Algorithm 1 may also converge to the unique* maximal *solution of the CARE even if a stabilizing solution does not exist (if the stabilizing solution exists, it coincides with the maximal one). In this case, convergence is usually only linear. Under some circumstances which are usually approached in later stages of the Newton iteration, it can be shown that taking a double Newton step, i.e., $X_{j+1} = X_j + 2N_j$, leads to the desired solution.*

With the above strategy, a double Newton step is always tried as this is nothing but setting $t_j = 2$. In that sense, Newton's method with exact line search includes the algorithm proposed in [29].

All the above considerations were employed in [9] for testing Newton's method with exact line search for all examples of the CARE benchmark collection [11]. All examples were solved successfully, employing the standard stabilization procedure from [53] for selecting X_0 and also using $X_0 = 0$ in case A is stable. Restarts were only encountered for two examples — in both cases due to the stagnation detected by (11).

In many applications in control theory, the solution of the CARE is used to compute the optimal control via the feedback law

$$u(t) = -R^{-1}B^T X_* E x(t).$$

Hence, instead of computing X_* we may in that case as well compute the product $X_E := X_* E$ directly. This can be used in Algorithm 1 as follows. Initialize $(X_E)_0$ such that $A - G(X_E)_0$ is stable. Solving the Lyapunov equation in Step 2. of the iteration for $(N_E)_j := N_j E$, we can compute $(X_E)_{j+1} = (X_E)_j + (N_E)_j$ and $A_j = A - G(X_E)_j$. Note that the residual $\mathcal{R}(X_j)$ and the matrix V_j needed for the line search procedure can also be computed using $(X_E)_j$ and $(N_E)_j$ without solving for X_j or N_j. We then get $\lim_{j\to\infty}(X_E)_j = X_E$. This approach saves at least solving a linear system with coefficient matrix E when solving the Lyapunov equations (see Section 3) and one matrix-matrix multiplication in each iteration step.

2.3 Condition estimation

Newton's method also provides an appropriate condition estimate for the CARE [4,15,34]. Consider the standard CARE

$$0 = \mathcal{R}(X) := Q + A^T X + XA - XGX. \tag{12}$$

Let $\hat{A}$, $\hat{G}$, and $\hat{Q}$, be matrices "near" A, G, and Q, respectively, and define $\Delta A = \hat{A} - A$, $\Delta G = \hat{G} - G$, and $\Delta Q = \hat{Q} - Q$. The condition number K_{CARE} of the standard CARE (12) is defined in [4,15,34] as

$$K_{\mathrm{CARE}} = \lim_{\delta \to 0} \sup \left\{ \frac{\|\Delta X\|}{\delta \|X\|} : \|\Delta A\| \leq \delta \|A\|, \|\Delta G\| \leq \delta \|G\|, \|\Delta Q\| \leq \delta \|Q\| \right\}.$$

Here, $\|\cdot\|$ denotes the 2-norm of a matrix. As computing this condition number is in general not possible, usually the approximate condition number for (12) given in [15] is used:

$$K_B := \frac{\|\Omega^{-1}\| \, \|Q\| + \|\Theta\| \, \|A\| + \|\Pi\| \, \|G\|}{\|X_*\|}, \tag{13}$$

where $A_c := A - GX_*$ and the linear operators Ω, Θ, Π mapping $\mathbb{R}^{n \times n}$ to $\mathbb{R}^{n \times n}$ are defined as

$$\begin{aligned} \Omega(Z) &= A_c^T Z + Z A_c \\ \Theta(Z) &= \Omega^{-1}(Z^T X_* + X_* Z) \\ \Pi(Z) &= \Omega^{-1}(X_* Z X_*) \end{aligned} \tag{14}$$

It can be shown that using the Frobenius norm, $(1/9) K_B \leq K_{\mathrm{CARE}} \leq 4 K_B$ [15] while for the matrix 2-norm, $(1/3) K_B \leq K_{\mathrm{CARE}} \leq K_B$ [34]. Computable upper and lower bounds K_U and K_L, respectively, for K_{CARE} and K_B are given in [34]: Let $\|\cdot\|$ denote the 2-norm and define

$$\begin{aligned} K_U &:= \frac{\|Z_0\|\|Q\| + 2\sqrt{\|Z_0\|\|Z_2\|}\|A\| + \|Z_2\|\|G\|}{\|X_*\|}, \\ K_L &:= \frac{\|Z_0\|\|Q\| + 2\|Z_1\|\|A\| + \|Z_2\|\|G\|}{\|X_*\|}, \end{aligned}$$

where Z_i, $i = 0, 1, 2$, are the solutions of the Lyapunov equations

$$(X_*)^i + (A - GX_*)^T Z_i + Z_i (A - GX_*) = 0, \quad i = 0, 1, 2. \tag{15}$$

The approximate condition number K_B is then bounded by $K_L \leq K_B \leq K_U$ which implies $\frac{1}{3} K_L \leq K_{\mathrm{CARE}} \leq K_U$. In case K_L is close to K_U, we have a highly accurate estimation of the condition number of the CARE.

We can use the solution of the CARE computed by Newton's method to estimate the condition of the standard CARE by means of K_L and K_U. Note that the equations (15) are of the same form as the Lyapunov equations that have to be solved during the Newton iteration and can therefore be solved by the same methods as suggested in Section 3.

The computationally expensive 2-norm can be estimated, e.g., as the geometric mean of the 1-norm and the infinity norm ($\|\cdot\| \approx \sqrt{\|\cdot\|_1 \|\cdot\|_\infty}$).

The condition number of the CARE in case $E \neq I_n$ has so far been addressed only in [4] where a condition number

$$K_{AL} = \frac{\mathrm{cond}\,(E)\,\mathrm{cond}\left(E^T\right)\,\|Q\|}{\|X_*\| \cdot \|E\| \cdot \|E^T\| \cdot \mathrm{sep}\,(A_c E^{-1}, -(A_c E^{-1})^T)},$$

with $\mathrm{cond}\,(E) := \|E\| \cdot \|E^{-1}\|$, is suggested. This condition number provides no information about the possible influence of perturbations in the coefficient matrices A and G and we therefore suggest an alternative here. For this purpose, we employ the ideas used in the standard case [15,34] and transform the CARE to the mathematical equivalent form

$$0 = \hat{Q} + \hat{A}^T X + X\hat{A} - XGX, \tag{16}$$

where $\hat{A} := AE^{-1}$, $\hat{Q} := E^{-T}QE^{-1}$. Note that X_* is the stabilizing solution of (16) if and only if it is the stabilizing solution of (1). Moreover, the coefficient G in the quadratic term is not changed by this transformation. Using the approximate condition number K_B we get that

$$K_B = \frac{\|\hat{\Omega}^{-1}\|\,\|\hat{Q}\| + \|\hat{\Theta}\|\,\|\hat{A}\| + \|\hat{\Pi}\|\,\|G\|}{\|X_*\|}. \tag{17}$$

$$\leq \frac{\|\hat{\Omega}^{-1}\|\,\|E^{-1}\|^2\,\|Q\| + \|\hat{\Theta}\|\,\|E^{-1}\|\,\|A\| + \|\hat{\Pi}\|\,\|G\|}{\|X_*\|} =: K_B^{(E)}. \tag{18}$$

Here, $\hat{\Omega}(Z) = (\hat{A} - GX_*)^T Z + Z(\hat{A} - GX_*)$, $\hat{\Theta}(Z) = \hat{\Omega}^{-1}(Z^T X_* + X_* Z)$, and $\hat{\Pi}(Z) = \hat{\Omega}^{-1}(X_* Z X_*)$. Now let Z_i, $i = 0, 1, 2$, be the solutions of the Lyapunov equations

$$(X_*)^i + (\hat{A} - GX_*)^T Z_i + Z_i(\hat{A} - GX_*) = 0, \quad i = 0, 1, 2. \tag{19}$$

The observations from [34] used to bound K_B by K_L and K_U, that is, $\|Z_0\| = \|\hat{\Omega}^{-1}\|$, $2\|Z_1\| \leq \|\hat{\Theta}\| \leq 2\sqrt{\|Z_0\|\,\|Z_2\|}$, and $\|Z_2\| = \|\hat{\Pi}\|$, lead to bounds $K_L^{(E)}$ and $K_U^{(E)}$ for the approximate condition number $K_B^{(E)}$. If

$$K_U^{(E)} := \frac{\|Z_0\|\,\|E^{-1}\|^2\,\|Q\| + 2\sqrt{\|Z_0\|\,\|Z_2\|}\,\|E^{-1}\|\,\|A\| + \|Z_2\|\,\|G\|}{\|X_*\|},$$

$$K_L^{(E)} := \frac{\|Z_0\|\,\|E^{-1}\|^2\,\|Q\| + 2\|Z_1\|\,\|E^{-1}\|\,\|A\| + \|Z_2\|\,\|G\|}{\|X_*\|},$$

then $K_L^{(E)} \leq K_B^{(E)} \leq K_U^{(E)}$. In order to compute $K_L^{(E)}$, $K_U^{(E)}$, we now have to determine (or estimate) the 2-norms of A, G, Q, E^{-1} and we have to solve the three Lyapunov equations in (20). Note that solving the equations in (19) is equivalent to solving the generalized Lyapunov equations

$$E^T(X_*)^i E + (A - GX_*E)^T Z_i E + E^T Z_i (A - GX_*E) = 0 \tag{20}$$

for $i = 0, 1, 2$. These have now the same form as the generalized Lyapunov equations that have to be solved in Step 2. of the Newton iteration in Algorithm 1. We will use this to design an almost optimal stopping criterion for Newton's method in the next section.

Remark 6 *a) Note that the condition number $K_B^{(E)}$ is only an upper bound for the real condition number of (1). This may be very conservative as it is based on transforming the CARE to a mathematical equivalent equation by inverting E. An estimate of the "real" condition number should better be based on the original formulation. This topic is under current investigation.*

b) It is pointed out in [34] that $2\sqrt{\|Z_0\|\,\|Z_2\|}$ may significantly over-estimate $\|\Theta\|$. An improved bound is therefore suggested in [34]. As this is computationally more involved and in most practical circumstances, K_L and K_U as introduced here come out to be very close, we only implemented the conservative bounds presented here.

2.4 Stopping criteria

Usually, convergence of Newton's method is based on either $\|\mathcal{R}(X_j)\| \leq tol\|X_j\|$ or $\|N_j\| \leq tol\|X_j\|$ for some user-defined tolerance threshold tol [4,44,54]. The first criterion is based on the scaled residual norm while N_j can be regarded as an estimate for the absolute error $X_* - X_j$ — at least in the final iterations. The threshold tol is usually of the form $c\varepsilon$ where c is some constant depending on n (usually $c = 10n$ or $c = 100n$) and ε is the machine precision.

We suggest here to split the iteration into two stages. In our implementations we employ a tolerance threshold $tol = c \cdot \sqrt{\varepsilon}$ and the stopping criterion based on the residual. Once this stopping criterion is satisfied, usually two additional iterations are enough to reach the attainable accuracy due to the quadratic convergence of Newton's method close to the solution. Moreover, this approach avoids possible stagnation of the method due to an un-attainable stopping criterion [9,12].

A more reliable stopping criterion can be based on the following observations. It is well-known [47] that the Newton iteration is able to improve the relative accuracy of an approximate solution X_j of the CARE as long as

$$\|X_* - X_j\| > \varepsilon K_{CARE}\|X_*\|.$$

This may be used as follows: once the relaxed stopping criterion suggested above is satisfied, we may in each subsequent iteration solve *four* Lyapunov equations with the same coefficient matrix A_j and the right hand sides $\mathcal{R}(X_j)$, $E^T X_j^i E$, $i = 0, 1, 2$, in order to obtain an estimate of the condition number K_B or $K_B^{(E)}$ of the CARE as well as the new Newton direction N_j. That is, we use A_j as an approximation to $A - GX_*E$ and X_j as an estimate to X_*. This is justified in the vicinity of the solution. We may then decide to stop the iteration if the current iterate satisfies

$$\|N_j\| \leq c\varepsilon K_{CARE}^{(j)}\|X_j\|. \tag{21}$$

This is based on estimating the current absolute error using N_j. Here, c is again some small constant, e.g., $c = 10n$ (For large n, this may be too conservative and $\sqrt{n}$ may yield better results). The stopping criterion (21) is justified using a result from [55] translated to the situation here: let $\|.\|$ denote either the 2-norm or the Frobenius norm and let $\Omega_j(Z) = (A - GX_jE)^T ZE + E^T Z(A - GX_jE)$ for $Z \in \mathbb{R}^{n \times n}$. If

$$4\|\Omega_j^{-1}\|\,\|\Omega_j^{-1}(\mathcal{R}(X_j))\|\|G\| < 1, \tag{22}$$

then

$$\begin{aligned}\frac{\|X_j - X_*\|}{\|X_j\|} &\leq \frac{2}{1 + \sqrt{1 - 4\|\Omega_j^{-1}\|\,\|\Omega_j^{-1}(\mathcal{R}(X_j))\|\,\|G\|}} \cdot \frac{\|\Omega_j^{-1}(\mathcal{R}(X_j))\|}{X_j} \\ &\leq \frac{2\|\Omega_j^{-1}(\mathcal{R}(X_j))\|}{\|X_j\|}.\end{aligned} \tag{23}$$

From Algorithm 1 it follows that $\Omega_j^{-1}(\mathcal{R}(X_j)) = -N_j$. (Note that X_j is stabilizing for all j by Theorem 3 and hence Ω_j is invertible.) Furthermore, close to the solution, $\|X_j\| \approx \|X_*\|$. Therefore, (23) yields the following bound for the relative error of the current iterate X_j:

$$\frac{\|X_j - X_*\|}{\|X_*\|} \approx \frac{\|X_j - X_*\|}{\|X_j\|} \leq \frac{2\|N_j\|}{X_j}. \tag{24}$$

In case we have estimated the condition number of the CARE using Ω_j and X_j as approximations to Ω and X_*, respectively, we can also use the tighter bound

$$\frac{\|X_j - X_*\|}{\|X_*\|} \approx \frac{\|X_j - X_*\|}{\|X_j\|} \leq \frac{2}{1 + \sqrt{1 - 4\|\Omega_j^{-1}\|\,\|N_j\|\,\|G\|}} \cdot \frac{\|N_j\|}{\|X_j\|}. \tag{25}$$

Note that $\|\Omega_j^{-1}\|$ is a by-product of the condition estimation using the above approach such that checking (22) and evaluating the bound for the relative error in (24) or (25) requires no additional computations. The quantity on the right hand side of inequality (24) or (25) can then also be used as a forward error estimate for the computed Riccati solution.

In Section 3.3 we show that the additional cost required to estimate the condition number of the CARE (which then gives the forward error bound and the stopping criterion basically for free) reduces significantly taking into account that all Lyapunov equations to be solved have the same coefficient matrices and only the right hand sides differ. Nevertheless, the refined stopping criterion involving condition and forward error estimation is only offered as an option due to the additional cost required. On the other hand, investing this additional cost not only yields a solution of highest attainable accuracy but also a measure for the reliability of the computed result via condition and error estimates.

3 Solving Lyapunov Equations with the Matrix Sign Function

We have observed that Newton's method and the condition estimation of the CARE require the solution of Lyapunov equations. Furthermore, the overall performance of Newton's method is determined by the performance of the Lyapunov solver. Therefore, we describe in this section several efficient algorithms for solving standard and generalized *stable* Lyapunov equations based on the matrix sign function.

3.1 The standard Lyapunov equation

Consider a matrix $Z \in \mathbb{R}^{n\times n}$, with no eigenvalues on the imaginary axis; let

$$Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1}$$

be its Jordan decomposition [27], where the Jordan blocks in $J^- \in \mathbb{R}^{k\times k}$ and $J^+ \in \mathbb{R}^{(n-k)\times(n-k)}$ contain, respectively, the eigenvalues of Z in the open left and right complex planes. The *matrix sign function* of Z is defined as

$$\operatorname{sign}(Z) \;:=\; S \begin{bmatrix} -I_k & 0 \\ 0 & I_{n-k} \end{bmatrix} S^{-1}. \tag{26}$$

Note that $\operatorname{sign}(Z)$ is unique and independent of the order of the eigenvalues in the Jordan decomposition of Z; see, e.g., [39]. Many other definitions of the sign function can be given; see [36] for an overview.

The matrix sign function has proved useful in many problems involving spectral decomposition as $(I_n - \operatorname{sign}(Z))/2$ defines the skew projector onto the stable Z-invariant subspace parallel to the unstable subspace. (By the *stable* invariant subspace of Z we denote the Z-invariant subspace corresponding to the eigenvalues of Z in the open left half plane.)

Applying Newton's method to $Z^2 = I_n$, where the starting point is chosen as Z, we obtain the Newton matrix sign function iteration

$$Z_0 \leftarrow Z, \qquad \texttt{FOR}\ j = 0,1,2,\ldots, \quad Z_{j+1} \leftarrow \frac{1}{2}(Z_j + Z_j^{-1}), \tag{27}$$

which converges globally and quadratically to $\operatorname{sign}(Z) = \lim_{j\to\infty} Z_j$ [49].

Although the convergence of the Newton matrix sign function iteration is globally quadratic, the initial convergence may be slow. Acceleration is possible, e.g., via *determinantal scaling* [16],

$$c_j = |\det(Z_j)|^{-\frac{1}{n}}, \quad Z_j \leftarrow c_j Z_j,$$

where $\det(Z_j)$ denotes the determinant of Z_j. Other acceleration schemes can be employed; see [5] for a comparison of these schemes.
Many iterative methods to compute the matrix sign function are known [35]. Among these, the *Halley iteration*

$$Z_0 \leftarrow Z, \quad \texttt{FOR}\ j = 0,1,2,\ldots, \quad Z_{j+1} \leftarrow Z_j(3I_n + Z_j^2)(I_n + 3Z_j^2)^{-1}, \tag{28}$$

and the *Newton-Schulz iteration*,

$$Z_0 \leftarrow Z, \qquad \texttt{FOR}\ j = 0,1,2,\ldots, \quad Z_{j+1} \leftarrow \frac{1}{2} Z_j(3I_n - Z_j^2), \tag{29}$$

are especially well suited to parallel computers with distributed memories as their dominant computational cost comes from matrix products. The higher computational cost of the Halley matrix sign function iteration can be balanced by its cubic convergence. Usually this is not the case, though, as the examples in Section 5 indicate. The Newton-Schulz matrix sign iteration requires no inverses but is only locally convergent. Its convergence can only be guaranteed in case $\|Z_0^2 - I_n\| < 1$ for some suitable norm. Therefore, the Newton-Schulz matrix sign function iteration has to be combined with some initial iteration to obtain a globally convergent "hybrid" algorithm.
Roberts [49] was the first to use the matrix sign function for solving Lyapunov (and Riccati) equations. In the proposed method, the solution of the Lyapunov equation (2) with $\tilde{E} = I_n$ and $\sigma(\tilde{A}) \subset \mathbb{C}^-$ is computed by applying the Newton iteration (27) to the Hamiltonian matrix $H = \left[\begin{smallmatrix} \tilde{A} & 0 \\ \tilde{Q} & -\tilde{A}^T \end{smallmatrix}\right]$ corresponding to (2). The solution matrix $\tilde{Y}$ can then be determined from the stable H–invariant subspace given by the range of the projector $(I_n - \operatorname{sign}(H))/2$. Roberts also shows in [49] that, when applied to H, the Newton matrix sign function iteration (27) can be simplified to

$$\begin{array}{l} \tilde{A}_0 \leftarrow \tilde{A}, \qquad \tilde{Q}_0 \leftarrow \tilde{Q}, \\ \texttt{FOR}\ j = 0,1,2,\ldots \\ \quad \tilde{A}_{j+1} \leftarrow \left(\tilde{A}_j + \tilde{A}_j^{-1}\right)/2, \\ \quad \tilde{Q}_{j+1} \leftarrow \left(\tilde{Q}_j + \tilde{A}_j^{-T}\tilde{Q}_j\tilde{A}_j^{-1}\right)/2, \end{array} \tag{30}$$

and that $\tilde{Y} = \frac{1}{2}\lim_{j\to\infty} \tilde{Q}_j$. The sequences for $\tilde{A}_j$ and $\tilde{Q}_j$ require $\mathcal{O}(6n^3)$ flops per iteration so that 5–6 iterations are as expensive as the Bartels–Stewart method [7].

The structure of H can also be exploited to obtain efficient variants for the Halley iteration (28),

```
$\tilde{A}_0 \leftarrow \tilde{A}, \qquad \tilde{Q}_0 \leftarrow \tilde{Q},$
FOR $j = 0, 1, 2, \ldots$
    $\tilde{A}_{j+1} \leftarrow \tilde{A}_j\,(3I_n + \tilde{A}_j^2)\,(I_n + 3\tilde{A}_j^2)^{-1},$                    (31)
    $\tilde{Q}_{j+1} \leftarrow \Big((\tilde{A}_j - 3\tilde{A}_{j+1})^T(\tilde{A}_j^T\tilde{Q}_j - \tilde{Q}_j\tilde{A}_j) +$
                  $+\, \tilde{Q}_j(3I_n + \tilde{A}_j^2)\Big)\,(I_n{+}3\tilde{A}_j^2)^{-1},$
```

and for the Newton-Schulz iteration (29),

```
$\tilde{A}_0 \leftarrow \tilde{A}, \qquad \tilde{Q}_0 \leftarrow \tilde{Q},$
FOR $j = 0, 1, 2, \ldots$
    $\tilde{A}_{j+1} \leftarrow \tilde{A}_j(3I_n - \tilde{A}_j^2)/2,$                    (32)
    $\tilde{Q}_{j+1} \leftarrow \Big(\tilde{Q}_j(3I_n - \tilde{A}_j^2) - \tilde{A}_j^T(\tilde{A}_j^T\tilde{Q}_j - \tilde{Q}_j\tilde{A}_j)\Big).$
```

The approximate computational costs per iteration for (31) and (32) are $\mathcal{O}(\frac{50}{3}n^3)$ and $\mathcal{O}(12n^3)$ flops, respectively. See [13] for details.

3.2 The generalized Lyapunov equation

Gardiner and Laub present in [21] a generalization of the matrix sign function for matrix pencils $L - \lambda M$, with L and M nonsingular. They consider the generalized Newton matrix sign function iteration (with determinantal scaling)

```
$L_0 \leftarrow L,,$
FOR $j = 0, 1, 2, \ldots$
    $c_j \leftarrow (|\det(L_j)|/|\det(M)|)^{\frac{1}{n}}$                    (33)
    $L_{j+1} \leftarrow (L_j + c_j^2 M L_j^{-1} M)/(2c_j).$
```

The iterates satisfy $\lim_{j\to\infty} L_j = M\text{sign}\,(M^{-1}L)$.
In case $\sigma\,(\tilde{E}, \tilde{A}) \subset \mathbb{C}^-$, we can apply iteration (33) to the matrix pencil

$$H - \lambda K := \begin{bmatrix} \tilde{A} & 0 \\ \tilde{Q} & -\tilde{A}^T \end{bmatrix} - \lambda \begin{bmatrix} \tilde{E} & 0 \\ 0 & \tilde{E}^T \end{bmatrix},$$

to obtain $\tilde{H}_\infty := K\text{sign}\,(K^{-1}H)$. The generalized Newton iteration can then be simplified [9,12] to

```
$\tilde{A}_0 \leftarrow \tilde{A}$, $\tilde{Q}_0 \leftarrow \tilde{Q}$,,
FOR $j = 0, 1, 2, \ldots$
    $\tilde{A}_{j+1} \leftarrow \left(\tilde{A}_j + \tilde{E}\tilde{A}_j^{-1}\tilde{E}\right)/2,$                (34)
    $\tilde{Q}_{j+1} \leftarrow \left(\tilde{Q}_j + \tilde{E}^T\tilde{A}_j^{-T}\tilde{Q}_j\tilde{A}_j^{-1}\tilde{E}\right)/2.$
```

with a cost of $\mathcal{O}(\frac{26}{3}n^3)$ flops per iteration, and the solution $\tilde{Y}$ is obtained from

$$\tilde{Y} \;=\; \frac{1}{2}\tilde{E}^{-T}\left(\lim_{j\to\infty}\tilde{Q}_j\right)\tilde{E}^{-1}.$$

The stopping criterion for the iteration (34) is based on $\lim_{j\to\infty}\tilde{A}_j = -\tilde{E}$ [12]. Similar stopping criteria can be employed in case $\tilde{E} = I_n$ for iterations (27), (28), and (29). For details of the iterative scheme (34), its implementation and numerical properties see [12].

3.3 Lyapunov equations with multiple right-hand sides

In order to estimate the condition number of the standard CARE by the method described in Section 2.3, we have to solve three Lyapunov matrix equations with the same coefficient matrix $(A - GX_*E)$. In case the Bartels-Stewart method is employed, this matrix is reduced to the real Schur form once and the three equations can be solved then by backward substitution. In case we use one of our matrix sign function solvers, the cost can also be reduced by using only one iteration with $A_0 = A - GX_*E$, and three iterations with $Q_0^{(i)} = E^T X_*^i E$, $i = 0, 1, 2$. In case this is used as suggested in Section 2.4, X_* is replaced by the current iterate X_j.
In general, solving several Lyapunov equations with the same coefficient matrix and different right-hand sides can be achieved in the same way. For r equations, one needs one iteration for the A_j's and r iterations for the solutions $Y^{(\ell)}$. If the right-hand side matrices are $Q^{(\ell)}$, $\ell = 1, \ldots, r$, then we have to initialize the iterations by $Q_0^{(\ell)} := Q^{(\ell)}$. A coarse-grain parallelism could be used here if we have r processors (or clusters of processors): compute A_j and broadcast it to the r logical processors, each of them holding one $Q_{j-1}^{(\ell)}$, the $Q_j^{(\ell)}$ can then be computed independent of each other.
This procedure is rather expensive with respect to workspace requirements as for each $Q^{(\ell)}$–iteration, in general an additional workspace of size n^2 is required. This workspace can be saved exploiting the symmetry of the $Q_j^{(\ell)}$'s. In that case, the computations have to be performed using only BLAS Level 2 subroutines. Despite saving the additional workspace and also reducing the cost of an update

$$Q_j^{(\ell)} \leftarrow Q_{j-1}^{(\ell)} + A_j^{-T} Q_{j-1}^{(\ell)} A_j^{-1}$$

from $4n^3$ flops to $3n^3$, the execution time on most processor architectures will be increased compared to forming the matrix products using the BLAS Level 3 matrix multiplication routines. So implementing this approach you either have to sacrifice speed of computations or workspace requirements.

3.4 Implementation details

In practice, the generalized Newton matrix sign function iteration for the Lyapunov equations often requires 7–12 iterations. The experimental results in [12] show that even with 20 iterations, a serial implementation of the generalized Lyapunov solver based on the matrix sign function is faster than the generalized Bartels-Stewart method ($\mathcal{O}(\frac{224}{3}n^3)$ flops, mainly for the QZ algorithm). This is due to the high efficiency of the computational kernels involved in the matrix sign function iterations. Moreover, we can expect even higher gains on parallel distributed architectures.
The cost of the Newton matrix sign function iteration for the generalized Lyapunov equation can be further reduced by computing an initial QR factorization of E, $E = Q_E R_E$. We may then set

$$A \leftarrow Q_E^T A \quad \text{and} \quad E \leftarrow Q_E^T E := R_E.$$

This only requires an additional update of the computed solution X by $X \leftarrow Q_E X Q_E^T$ (step (3) in Algorithm 2). With this transformation, the matrix multiplications with E in Algorithm 2 only require multiplication by a triangular matrix and the linear systems to be solved in order to obtain X in the last stage are triangular systems. Nevertheless, the experimental studies in [12] report higher execution times of this approach due to the lower efficient implementation of the computational kernels for triangular matrix operations.
As in Newton's method, in our implementations of the matrix sign function iterations we employ a tolerance threshold $tol = c \cdot \sqrt{\varepsilon}$. Once the stopping criterion is satisfied, we perform two additional iterations. Due to the quadratic convergence of the iterations (cubic for Halley), this is usually enough to reach the attainable accuracy and avoids possible stagnation of the iterations [9,12].
The convergence of the Newton-Schulz iteration is only guaranteed in case $||H_k^2 - I_{2n}|| < 1$. We have therefore developed a hybrid matrix sign function iteration solver composed of the Newton and Newton-Schulz iterations. Specifically, the Newton iteration is applied until $||H_{k+1} - H_k|| < 1$ and, from then on, the Newton-Schulz iteration is used. This "switching" criterion shows a good behavior in practice. Other less effective criteria have also been tested. For instance, the criterion used in [6], $||H_{k+1} - H_k|| < \sqrt{2n}$, does not guarantee the convergence of the Newton-Schulz iteration; $||H_k - I_{2n}||||H_k + I_{2n}|| < 1$ tends to switch too late, and $||H_k^2 - I_{2n}|| < 1$ is computationally too expensive.

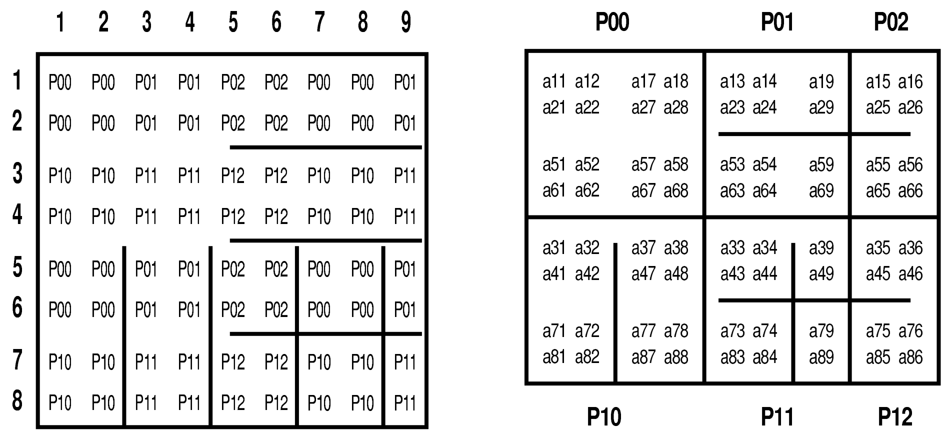
4 Parallelization Issues and Performance Analysis

Our parallel algorithms are implemented using ScaLAPACK (scalable linear algebra package) [14]. This is a public-domain parallel library for MIMD computers which provides scalable parallel distributed subroutines for many matrix algebra kernels available in LAPACK [2]. The ScaLAPACK library employs BLAS and LAPACK for serial computations, PB-BLAS (parallel block BLAS) for parallel basic matrix algebra computations, and BLACS (basic linear algebra communication subprograms) for communication.

The efficiency of the ScaLAPACK kernels depends on the efficiency of the underlying computational BLAS/PB-BLAS routines and the communication BLACS library. BLACS can be used on any machine which supports either PVM [25] or MPI [28], thus providing a highly portable environment.

In ScaLAPACK, the user is responsible for distributing the data among the processes. Access to data stored in a different process must be explicitly requested and provided via message-passing.

The implementation of ScaLAPACK employs a block-cyclic distribution scheme [14], which is mapped into a logical $p_r \times p_c$ grid of processes. Each process owns a collection of (`MB` $\times$ `NB`) blocks, which are locally and contiguously stored in a two-dimensional array in "column-major" order (see Figure 2).



1 2 3 4 5 6 7 8 9
1 P00 P00 P01 P01 P02 P02 P00 P00 P01
2 P00 P00 P01 P01 P02 P02 P00 P00 P01
3 P10 P10 P11 P11 P12 P12 P10 P10 P11
4 P10 P10 P11 P11 P12 P12 P10 P10 P11
5 P00 P00 P01 P01 P02 P02 P00 P00 P01
6 P00 P00 P01 P01 P02 P02 P00 P00 P01
7 P10 P10 P11 P11 P12 P12 P10 P10 P11
8 P10 P10 P11 P11 P12 P12 P10 P10 P11
P00
P01
P02
a11 a12 a17 a18 a13 a14 a19 a15 a16
a21 a22 a27 a28 a23 a24 a29 a25 a26
a51 a52 a57 a58 a53 a54 a59 a55 a56
a61 a62 a67 a68 a63 a64 a69 a65 a66
a31 a32 a37 a38 a33 a34 a39 a35 a36
a41 a42 a47 a48 a43 a44 a49 a45 a46
a71 a72 a77 a78 a73 a74 a79 a75 a76
a81 a82 a87 a88 a83 a84 a89 a85 a86
P10
P11
P12

Fig. 2. Block-cyclic data storage scheme of an 8×9 matrix, $p_r \times p_c$=2×3 and `MB`$\times$`NB`=2×2.

For scalability purposes, we only employ square topologies ($p_r = p_c$) with each process mapped onto a different processor. The following parameters are used:

- n: Size of the problem.
- $p = \sqrt{p} \times \sqrt{p}$: Dimension of the square grid of processors.
- τ: Execution time required for a double-precision floating-point arithmetic operation.
- α (latency): Time required to communicate a zero-length message between two processors.
- β (reciprocal of bandwidth): Time required to communicate a double-precision floating-point number between two processors.

For simplicity, we assume here that the time required to perform any floating-point arithmetic operation is constant. In practice this execution time depends strongly on several architecture-dependent factors. We also assume a linear model for communication (i.e., the time required to communicate an message of length m is $\alpha + \beta m$).

We have employed the following (double precision) routines from ScaLAPACK [14] in our CARE solvers:

- `PDGETRF`: LU factorization with partial pivoting.
- `PDTRSV`: Solution of a triangular linear system.
- `PDGETRI`: Matrix inversion using the LU factors.
- `PDGEMM`: Matrix product.

Following the performance model in [6], we present in Table 1 approximate computation and communication costs for these routines (lower order expression have been neglected).

Parallel routine	Computation cost	Communication cost: Latency	Communication cost: Bandwidth^{-1}
`PDGETRF`	$\frac{2}{3} \times \frac{n^3}{p}\, \tau$	$(6 + \log p) n \times \alpha$	$(3 + \frac{1}{4}\log p) \times \frac{n^2}{\sqrt{p}}\, \beta$
`PDTRSV`	$\frac{n^3}{p}\, \tau$	$n \times \alpha$	$(1 + \frac{3}{4}\log p) \times \frac{n^2}{\sqrt{p}}\, \beta$
`PDGETRI`	$\frac{4}{3} \times \frac{n^3}{p}\, \tau$	$2n \times \alpha$	$(2 + \frac{3}{2}\log p) \times \frac{n^2}{\sqrt{p}}\, \beta$
`PDGEMM`	$2 \times \frac{n^3}{p}\, \tau$	$(1 + \frac{1}{2}\log p)\sqrt{p} \times \alpha$	$(1 + \frac{1}{2}\log p) \times \frac{n^2}{\sqrt{p}}\, \beta$

Table 1
Approximate computation and communication costs for several parallel routines in ScaLAPACK.

The number of calls to these subroutines in our CARE solvers depends on the number of iterations of Newton's method, the matrix sign function iterative scheme employed for solving the Lyapunov equation (Newton, Halley, Newton-Schulz, or generalized Newton), and the number of iterations of the matrix sign function for the Lyapunov solver.

In Table 2 we show the number of subroutine calls required by the Lyapunov solver employed in a single iteration of Newton's method for the CARE. In the table, "iter" stands for the number of iterations necessary for the convergence of the corresponding matrix sign function iteration for the Lyapunov equation.

From the data in Tables 1 and 2 we construct the theoretical performance model for our parallel CARE solvers in Tables 3 and 4. Note that the performance model only refers to a single iteration of Newton's method.

We have observed moderate differences between the theoretical results obtained from our performance model and the experimental results; these differences are due to simplifications and approximations in the theoretical model; e.g., constant cost for any type of floating-point arithmetic operation, linear model for communications, imbalance cost and lower order terms in computations/communications are neglected, etc.

Iteration for Lyapunov eq.	PDGETRF	PDTRSV	PDGETRI	PDGEMM
Newton	iter	$4 \times$ iter	iter	7
Halley	iter	$4 \times$ iter	–	$7 + 6 \times$ iter
Newton-Schulz	–	–	–	$7 + 6 \times$ iter
Generalized Newton	$2 +$ iter	$4 + 2 \times$ iter	–	$8 + 3 \times$ iter

Table 2
Number of calls to different parallel routines from ScaLAPACK required by a single iteration of Newton's method for the CARE.

Iteration for Lyapunov eq.	Computation cost $\times \frac{n^3}{p}\,\tau$
Newton	$14 + 6 \times$ iter
Halley	$14 + \frac{50}{3} \times$ iter
Newton-Schulz	$14 + 12 \times$ iter
Generalized Newton	$\frac{64}{3} + \frac{26}{3} \times$ iter

Table 3
Computation cost of a single iteration of Newton's method for the CARE.

Iteration for	Communication cost	
Lyapunov eq.	Latency $\times\, \alpha$	Bandwidth$^{-1} \times \frac{n^2}{\sqrt{p}}\,\beta$
Newton	$7(1 + \frac{1}{2}\log p)\sqrt{p}\, +$ $(12 + \log p)n \times$ iter	$7(1 + \frac{1}{2}\log p)\, +$ $(9 + \frac{19}{4}\log p) \times$ iter
Halley	$(1 + \frac{1}{2}\log p)(7 + 6 \times \text{iter})\sqrt{p}\, +$ $(10 + \log p)n \times$ iter	$(1 + \frac{1}{2}\log p)(7 + 6 \times \text{iter})\, +$ $(7 + \frac{13}{4}\log p) \times$ iter
Newton-Schulz	$(1 + \frac{1}{2}\log p)(7 + 6 \times \text{iter})\sqrt{p}$	$(1 + \frac{1}{2}\log p)(7 + 6 \times \text{iter})$
Generalized Newton	$(1 + \frac{1}{2}\log p)(8 + 3 \times \text{iter})\sqrt{p}\, +$ $(8 + \log p)(2 + \text{iter})n$	$(1 + \frac{1}{2}\log p)(8 + 3 \times \text{iter})\, +$ $(5 + \frac{7}{4}\log p)(2 + \text{iter})$

Table 4
Communication cost of a single iteration of Newton's method for the CARE.

5 Numerical Experiments

In this section we compare the performance of our CARE solvers for several benchmark examples. Some of these problems (Examples 7, 8, and 9) are generated with the Fortran routine `carex.f` [11]. For comparisons of Newton's method with and without line search see [9,8,10].

The experiments were performed using Fortran 77 and IEEE double-precision arithmetic ($\varepsilon \approx 2.2 \times 10^{-16}$) on an IBM SP2 platform. We made use of the vendor-supplied BLAS (*essl*), and the LAPACK library [2]. BLACS are installed on top of MPI.

5.1 Serial performance of CARE solvers

In the following examples we compare the performance of the serial CARE solvers based on Newton's method. We denote the solvers for the standard CARE ($E = I_n$) by `DGECRxx` and those for the generalized CARE ($E \neq I_n$) by `DGGCRxx`. Unless otherwise explicitly stated, all the methods employ the exact line search acceleration. The suffix `-xx` identifies the Lyapunov equation solver employed in the algorithm as follows:

- `DGECRBS` and `DGGCRBS`: The Bartels-Stewart method.
- `DGECRNE` and `DGGCRNE`: The Newton iteration.
- `DGECRHA`: The Halley iteration.
- `DGECRNS`: The hybrid iteration (Newton iteration followed by Newton-Schulz iteration).

In case A is stable, we use $X_0 = 0_n$ as the starting guess in our CARE solvers; otherwise, we use an initial solution $X_0 = \tilde{X}$ computed by means of the Schur vector method (we use in this case our routines `DGECRSV` and `DGGCRSV` [48] for the standard and the generalized CARE, respectively). The reported test results were obtained using the simple two-stage stopping criterion suggested in Section 2.4. That is, the tolerance threshold is set to $tol = 10 \cdot n \cdot \sqrt{\varepsilon}$ in the stopping criterion of both the matrix sign function iterations and Newton's method for the CARE and two additional iteration steps are performed once the criterion is satisfied. The figures report the time required by Newton's method to refine the initial solution X_0 to "maximum" accuracy.

Example 7 *This example describes a mathematical model of position and velocity control of a string of high-speed vehicles [42]. The condition number of the example only grows very slowly with n.*

The left-hand plot in Figure 3 reports the execution time of the CARE solvers with $X_0 = \tilde{X}$ (initial solution by the Schur vector method) and n=249, 499, 749, and 999. Newton's method requires only three iterations to converge in these cases ($\|\mathcal{R}(X_0)\|_F \approx 1.0 \times 10^{-10}$ and $\|\mathcal{R}(X_3)\|_F \approx 1.0 \times 10^{-14}$). (Actually, two iterations of Newton's methods are enough to reach the maximum accuracy but, as we perform two iterations after the convergence criterion is satisfied, we get an overall number of three iterations.)

On average, solving the Lyapunov equations in this example requires 11 iterations of the Newton matrix sign function iteration, 8+5 iterations of the hybrid (Newton+Newton-Schulz) matrix sign function iteration, and 6 iterations of the Halley matrix sign function iteration.

Example 8 *The system matrices and the solution matrices of this example are circulant [42]. The exact solution is known a priori and the condition number is small and independent of n.*

The right-hand plot in Figure 3 reports the execution time of the CARE solvers with $X_0 = \tilde{X}$ and n=250, 500, 750, and 1000. Three iterations of Newton's method are enough to obtain $\|X_* - X_3\|_F / \|X_*\|_F \approx 1.0 \times 10^{-12}$ ($\|\mathcal{R}(X_0)\|_F \approx 1.0 \times 10^{-12}$ and $\|\mathcal{R}(X_3)\|_F \approx 1.0 \times 10^{-15}$). Two iterations of Newton's methods *are enough to reach the maximum accuracy. The third iteration is due to our relaxed stopping criterion.*

On average, solving the Lyapunov equations in this example requires 6 iterations of the Newton matrix sign function iteration, 3+3 iterations of the the hybrid (Newton+Newton-Schulz) matrix sign function iteration, and 5 iterations of the the Halley matrix sign function iteration.

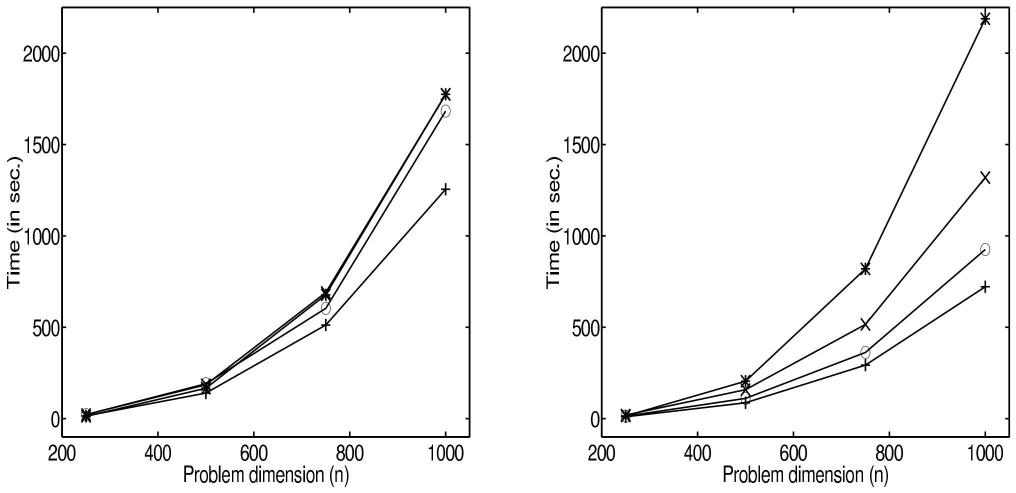


Fig. 3. Execution time of Newton's method for Examples 7 (left) and 3 (right) on the IBM SP2. Legend: symbols "—∗—" for DGECRBS, "—·+·—" for DGECRNE, "···o···" for DGECRNS, and "– – × – –" for DGECRHA.

Examples 7 and 8 show that, in the standard case, the best solver is the one based on the Newton iteration for the Lyapunov equation (DGECRNE). For medium and large-size problems ($n \geq 500$), the solver based on the hybrid iteration (DGECRNS) is also more efficient than the one based on the Bartels-Stewart method (DGECRBS). In Example 7, the cubic convergence of the solver based on the Halley iteration (DGECRHA) is overrun by its higher computational cost.

Example 9 *This is Example 18 from [11], parameterized by the following seven arguments:* $\alpha = 10^{-2}$, $\beta = 1$, $\gamma = 1$, $\beta_1 = 0.2$, $\beta_2 = 0.3$, $\gamma_1 = 0.2$, *and* $\gamma_2 = 0.3$. *We use our generalized CARE solvers on the generalized problem; no attempt is made to exploit the special structure of this problem.*

The left-hand plot in Figure 4 reports the execution time of the CARE solvers for n=250, 500, 750, and 1000. The solution computed by the generalized Schur vector method DGGCRSV *satisfies* $||\mathcal{R}\left(\tilde{X}\right)||_F \approx 1.0 \times 10^{-8}$. *Starting the iteration at* $X_0 = 0_n$, *Newton's method requires five iterations to converge for this problem (in both algorithms,* DGGCRBS *and* DGGCRNE, $||\mathcal{R}\left(X_5\right)||_F \approx 1.0 \times 10^{-12}$*). The execution time of Newton's method based on solving the Lyapunov equations with the Bartels-Stewart method is close to that of the generalized Schur vector method. Newton's method based on the generalized Newton matrix sign function iteration is far more efficient as this approach is up to 5 times* *as fast as the other two methods. Also note that in order to refine the solution computed by the Schur vector method to the maximal attainable accuracy, 1–2 iterations of Newton's method are necessary, increasing the overall cost of the Schur vector method even more.*

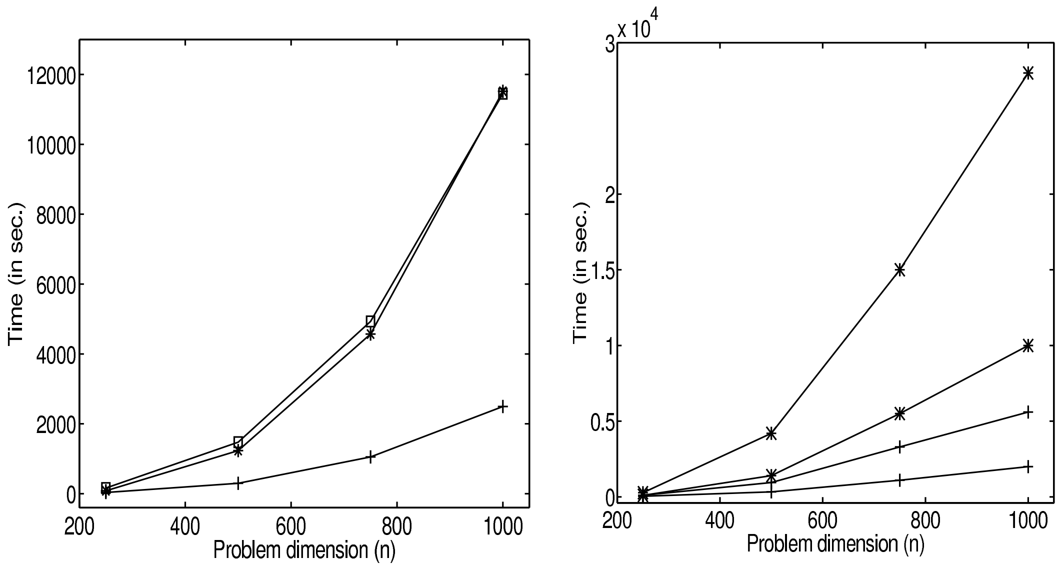


Fig. 4. Execution time of Newton's method for Example 9 on the IBM SP2. Legend: symbols "–□–" for DGGCRSV, "$-*-$" for DGGCRBS, and "$-\cdot+\cdot-$" for DGGCRNE. In the right-hand plot, "$\cdots*\cdots$ for DGGCRBS without exact line search, and "$\cdots+\cdots$" for DGGCRNE without exact line search.

This example also illustrates the benefits of using the exact line search. We modify the parameters of the problem to $\alpha = 10^{-4}$, $\beta = 1$, $\gamma = 100$, $\beta_1 = 0.1$, $\beta_2 = 0.5$, $\gamma_1 = 0.2$, and $\gamma_2 = 0.3$. The right-hand plot in Figure 4 reports the execution time of the CARE solvers with/without exact line search for n=250, 500, 750, and 1000. Starting the iteration at $X_0 = 0_n$, the use of the exact line search in Newton's method reduces the number of iterations from 17 to 6 ($\|\mathcal{R}(X_6)\|_F \approx 1.0 \times 10^{-13}$).

Example 10 *We generate random matrix pairs*

$$A = V_n \text{diag}(\alpha_1, \dots, \alpha_n) W_n, \qquad E = V_n W_n,$$

where the scalars $\alpha_1, \dots, \alpha_n$ are uniformly distributed in $[-10, 0)$, and W_n is an $n \times n$ lower triangular matrix. In the standard case, $V_n = W_n^{-1}$ so that $E = I_n$; in the generalized case, V_n is an $n \times n$ matrix with unit entries on and below the anti-diagonal and all other entries equal to zero. Then, we construct two random $n \times n$ symmetric positive semidefinite matrices, X_* and G, with $\|X_*\|_F = \|G\|_F = 1$. Matrix Q is then constructed as $Q = -(A^T X_* E + E^T X_* A - E^T X_* G X_* E)$.

Figure 5 shows the execution time of the CARE solvers with $\tilde{X} = 0_n$ and n=250, 500, 750, and 1000. In this example we only report the results of the first iteration of Newton's method. The number of iterations of Newton's *method should be roughly the same for all CARE solvers since they only differ in the Lyapunov solver that is employed.*

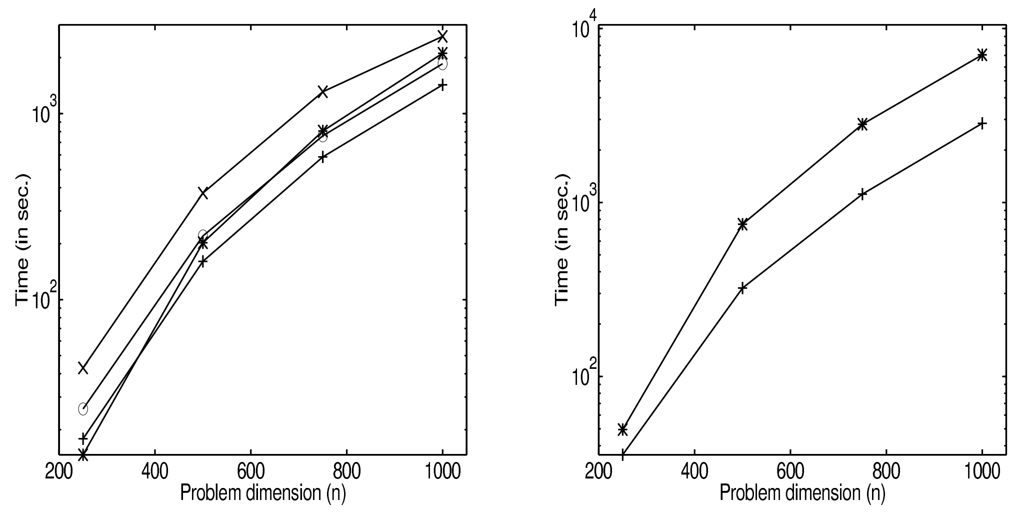


Fig. 5. Execution time of one iteration of Newton's method for the standard case (left) and the generalized case (right) of Example 10 on the IBM SP2. Legend: symbols "$-*-$" for DGECRBS and DGGCRBS, "$-\cdot+\cdot-$" for DGECRNE and DGGCRNE, "$\cdots\text{o}\cdots$" for DGECRNS, and "$--\times--$" for DGECRHA.

*This example shows that both in the standard and generalized case, the best solvers are those based on the Newton matrix sign function iteration (*DGECRNE *and* DGGCRNE*). In the standard case solver* DGECRNS *is also more efficient than solver* DGECRBS *for medium and large-size problems, while the most expensive standard CARE solver is algorithm* DGECRHA *in all cases.*

5.2 Parallel performance of the CARE solvers

The IBM SP2 architecture consists of 80 RS6000 SP2 THIN nodes at 120 MHz, and 256 MBytes RAM per processor. Internally, the nodes are connected by a TB3 high performance switch. The latency is 31 microseconds and the bandwidth is about 90 MBytes/sec. We use BLACS (on top of MPI), and ScaLAPACK [2,14] to ensure the portability of the algorithms to other serial and parallel architectures.

In Figure 6 we analyze the performance of the parallel implementations of the CARE solvers. The compared codes are PDGECRNE, PDGECRNS, PDGECRHA, and PDGGCRNE, based on solving the Lyapunov equations by the Newton, hybrid, Halley, and generalized Newton iteration. Parallel solvers for the Schur vector method and Newton's method based on Bartels-Stewart algorithm are not implemented due to the lack of the necessary matrix algebra kernels in the current version of the ScaLAPACK library.

We only report the Mflop rate (millions of flops per second) per node on square grids of 1, 4, 9, and 16 nodes of one iteration of Newton's method. This ratio is independent of the number of iterations and is computed by dividing the number of flops required for each algorithm by the product of the runtime and the number of nodes.

To analyze the *scalability* of the solvers, we fix the dimension of the problem per node in the experiment to $n/\sqrt{p} = 500$, 750, and 1000. The figures report a high scalability of our solvers as there is only a slight decrease in the Mflop rate when the number of processors is increased from 4 to 16. This result agrees with the scalability of the basic building blocks involved in the CARE solvers (matrix product, LU factorization, triangular linear systems, etc.) In some cases, increasing the ratio $n/\sqrt{p}$ from 750 to 1000 reduces the performance of the algorithms. Further experiments showed that this is due to the cache size of this architecture.

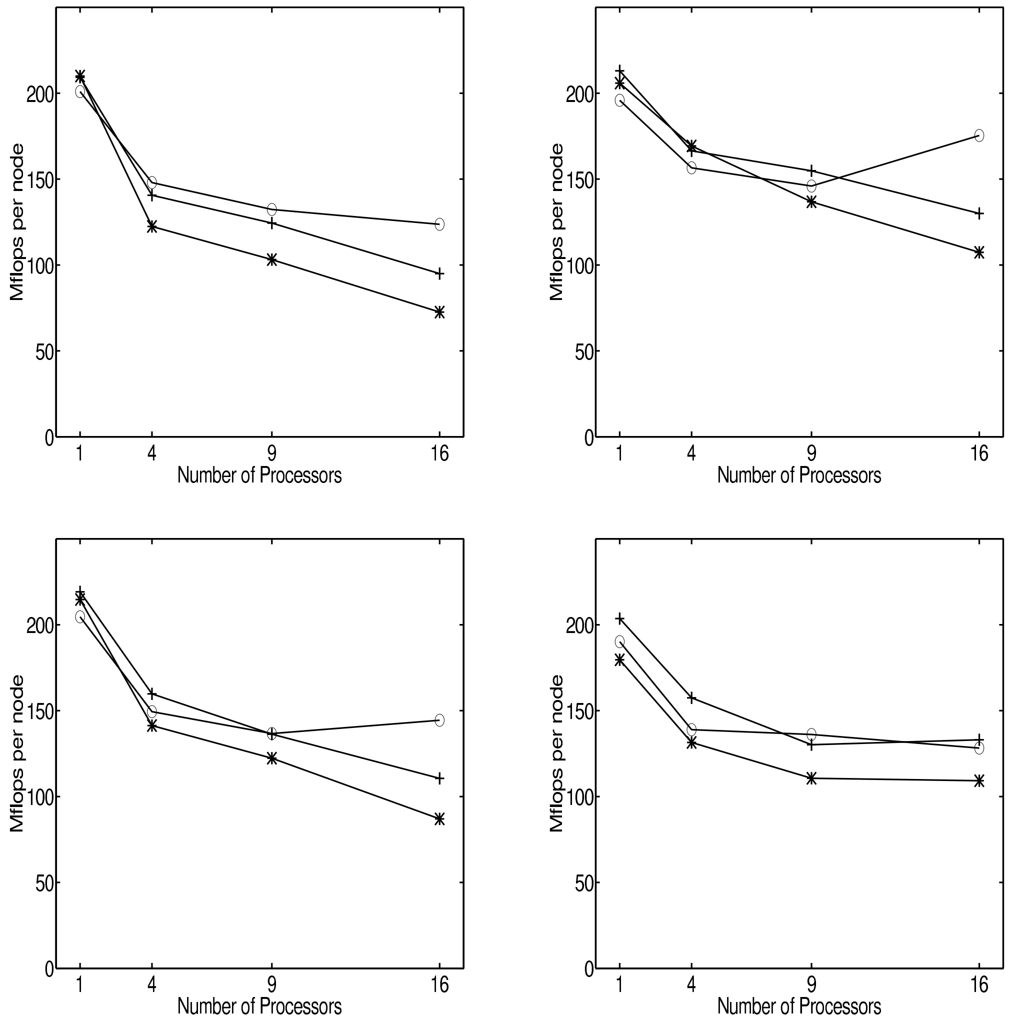


Fig. 6. Mflop rate per node of the parallel solvers `PDGECRNE` (upper left) and `PDGECRHA` (upper right) `PDGECRNS` (lower left) and `PDGGCRNE` (lower right) on the IBM SP2. Legend: symbols "$\cdots * \cdots$" for $n/\sqrt{p} = 500$, "$\cdots + \cdots$" for $n/\sqrt{p} = 750$, and "$\cdots \circ \cdots$" for $n/\sqrt{p} = 1000$.

Consider $T(n,p)$ as the execution time of an algorithm for solving a problem of size n using p processors. The *speed-up* of the algorithm is defined as the ratio $S_p(n,p) = T(n,1)/T(n,p)$. We are usually interested in the speed-up of an algorithm for large n; however, due to memory size restrictions on the IBM SP2, we can only solve problems with $n/\sqrt{p}$ up to 1000 and therefore we can only compute the speed-up for problems of size n=1000. We will therefore compute an approximation of this measure as follows. Assume that $T(n,p) = kn^3/p$ for some constant k; we have then $T(n\sqrt{p},p) = T(n,p) \cdot p\sqrt{p}$ and we can compute an estimated speed-up as

$$\tilde{S}_p(n,p) = \frac{T(n,1)}{T(n\sqrt{p},p)} \cdot p\sqrt{p}.$$

In case we keep $n/\sqrt{p}$ fixed at 1000 we obtain the estimated speed-ups in Table 5. The results show that parallel solvers based on Halley and Newton-Schulz matrix sign function iterations present higher speed-ups. This is due to the higher number of involved matrix products as these are specially efficient on parallel distributed architectures.

Notice that a higher Mflop rate or a higher speed-up does not necessarily imply a more efficient CARE solver since the efficiency also depends on the number of iterations of the matrix sign function iterative schemes required to solve each Lyapunov equation.

Algorithm	p=4	p=9	p=16
DGECRNE	2.96	5.94	9.84
DGECRHA	3.20	6.69	14.35
DGECRNS	2.92	6.00	11.29
DGGCRNE	2.90	6.44	10.77

Table 5
Estimated speed-ups $\tilde{S}_p(n,p)$ of the parallel CARE solvers ($n/\sqrt{p}$=1000).

6 Concluding Remarks

We have analyzed the convergence theory and implementation of Newton's method with exact line search for solving standard and generalized CAREs. Our algorithms employ matrix sign function iterations (e.g., Newton, Newton-Schulz, and Halley) for solving the Lyapunov equations arising in each iteration step of Newton's method. Based on known condition estimates for the standard CARE and their extension to the generalized case we propose a reliable stopping criterion. From this we also obtain a forward error bound for the relative error of the computed solution.

Unlike the Bartels-Stewart method which is based on the QR/QZ algorithm, our parallel implementations are composed of medium-grain scalable computational kernels. The experimental results on serial computers and an IBM SP2 parallel distributed platform show high accuracy, performance, and scalability of our solvers.

Acknowledgement

We thank the Mathematics and Computer Science Division at Argonne National Laboratory for the use of the IBM SP2.

References

[1] B. D. O. Anderson and J. B. Moore. *Linear Optimal Control.* Prentice-Hall, Englewood Cliffs, NJ, 1971.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, second edition, 1995.

[3] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific J. Math.*, 16:1–3, 1966.

[4] W.F. Arnold, III and A.J. Laub. Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proc. IEEE*, 72:1746–1754, 1984.

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

[6] Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, and K. Stanley. The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers. *SIAM J. Sci. Comput.*, 18:1446–1461, 1997.

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

[8] P. Benner. Numerical solution of special algebraic Riccati equations via an exact line search method. In *Proc. European Control Conf. ECC 97,* Paper 786. BELWARE Information Technology, Waterloo, Belgium, 1997. CD-ROM.

[9] P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems.* Logos–Verlag, Berlin, Germany, 1997. *Also:* Dissertation, Fakultät für Mathematik, TU Chemnitz–Zwickau, 1997.

[10] P. Benner and R. Byers. An exact line search method for solving generalized continuous-time algebraic Riccati equations. *IEEE Trans. Automat. Control,* 43(1):101–107, 1998.

[11] P. Benner, A.J. Laub, and V. Mehrmann. A collection of benchmark examples for the numerical solution of algebraic Riccati equations I: Continuous-time case. Technical Report SPC 95_22, Fakultät für Mathematik, TU Chemnitz–Zwickau, 09107 Chemnitz, FRG, 1995. Available from `http://www.tu-chemnitz.de/sfb393/spc95pr.html`.

[12] P. Benner and E.S. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. Technical Report SFB393/97-23, Fakultät für Mathematik, TU Chemnitz, 09107 Chemnitz, FRG, 1997. Available from `http://www.tu-chemnitz.de/sfb393/sfb97pr.html`.

[13] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear matrix equations via rational iterative schemes. In preparation.

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

[15] R. Byers. Numerical condition of the algebraic Riccati equation. *Contemp. Math.*, 47:35–49, 1985.

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

[17] J. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[18] J.J. Dongarra, A. Sameh, and D. Sorensen. Implementation of some concurrent algorithms for matrix factorization. *Parallel Comput.*, 3:25–34, 1986.

[19] Z. Gajić and X. Shen. *Parallel Algorithms for Optimal Control of Large Scale Linear Systems.* Springer-Verlag, London, GB, 1994.

[20] J. D. Gardiner. Stabilizing control for second-order models and positive real systems. *AIAA J. Guidance, Dynamics and Control*, 15(1):280–282, 1992.

[21] J.D. Gardiner and A.J. Laub. A generalization of the matrix-sign-function solution for algebraic Riccati equations. *Internat. J. Control*, 44:823–832, 1986.

[22] J.D. Gardiner and A.J. Laub. Solving the algebraic Riccati equation on a hypercube multiprocessor. In G. Fox, editor, *Hypercube Concurrent Computers and Applications, Vol. II*, pages 1562–1568. ACM Press, New York, 1988.

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

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

[25] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Network Parallel Computing.* MIT Press, 1994.

[26] A.R. Ghavimi, C. Kenney, and A.J. Laub. Local convergence analysis of conjugate gradient methods for solving algebraic Riccati equations. *IEEE Trans. Automat. Control*, 37:1062–1067, 1992.

[27] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, second edition, 1989.

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

[29] C.-H. Guo and P. Lancaster. Analysis and modification of Newton's method for algebraic Riccati equations. *Math. Comp.*, 67:1089–1105, 1998.

[30] S.J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[31] G. Henry and R. van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality. *SIAM J. Sci. Comput.*, to appear.

[32] G. Henry, D.S. Watkins, and J.J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. Technical Report LAPACK Working Note 121, University of Tennessee at Knoxville, 1997.

[33] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, Philadelphia, PA, 1995.

[34] C. Kenney and G. Hewer. The sensitivity of the algebraic and differential Riccati equations. *SIAM J. Cont. Optim.*, 28:50–69, 1990.

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

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

[37] C. Kenney, A.J. Laub, and M. Wette. Error bounds for Newton refinement of solutions to algebraic Riccati equations. *Math. Control, Signals, Sys.*, 3:211–224, 1990.

[38] D. L. Kleinman. On an iterative technique for Riccati equation computations. *IEEE Trans. Automat. Control*, AC-13:114–115, 1968.

[39] P. Lancaster and L. Rodman. *The Algebraic Riccati Equation.* Oxford University Press, Oxford, 1995.

[40] P. Lancaster and M. Tismenetsky. *The Theory of Matrices.* Academic Press, Orlando, 2nd edition, 1985.

[41] I. Lasiecka and R. Triggiani. *Differential and Algebraic Riccati Equations with Application to Boundary/Point Control Problems: Continuous Theory and Approximation Theory.* Number 164 in Lecture Notes in Control and Information Sciences. Springer-Verlag, Berlin, 1991.

[42] A.J. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Trans. Automat. Control*, AC-24:913–921, 1979.

[43] A.J. Laub and J.D. Gardiner. Hypercube implementation of some parallel algorithms in control. In M.J. Denham and A.J. Laub, editors, *Advanced Computing Concepts and Techniques in Control Engineering*, pages 361–390. Springer-Verlag, Berlin, 1988.

[44] V. Mehrmann. *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution.* Number 163 in Lecture Notes in Control and Information Sciences. Springer-Verlag, Heidelberg, July 1991.

[45] P. Pandey, C. Kenney, and A.J. Laub. A parallel algorithm for the matrix sign function. *Int. J. High Speed Computing*, 2:181–191, 1990.

[46] T. Penzl. Numerical solution of generalized Lyapunov equations. *Adv. Comp. Math.*, 8:33–48, 1997.

[47] P.H. Petkov, N.D. Christov, and M.M. Konstantinov. *Computational Methods for Linear Control Systems.* Prentice-Hall, Hertfordshire, UK, 1991.

[48] E.S. Quintana-Ortí. *Algoritmos Paralelos Para Resolver Ecuaciones Matriciales de Riccati en Problemas de Control.* PhD thesis, Universidad Politécnica de Valencia, 1996.

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

[50] I.G. Rosen and C. Wang. A multi–level technique for the approximate solution of operator Lyapunov and algebraic Riccati equations. *SIAM J. Numer. Anal.*, 32(2):514–541, 1995.

[51] M.G. Safonov and R.Y. Chiang. Model reduction for robust control: A Schur relative error method. *Int. J. Adapt. Cont. and Sign. Proc.*, 2:259–272, 1988.

[52] G. Schelfhout. *Model Reduction for Control Design.* PhD thesis, Dept. Electrical Engineering, KU Leuven, 3001 Leuven–Heverlee, Belgium, 1996.

[53] V. Sima. An efficient Schur method to solve the stabilization problem. *IEEE Trans. Automat. Control*, AC-26:724–725, 1981.

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

[55] J.-G. Sun. Residual bounds of approximate solutions of the algebraic Riccati equation. *Numer. Math.*, 76:249–263, 1997.

[56] A. Varga. On stabilization methods of descriptor systems. *Sys. Control Lett.*, 24:133–138, 1995.

[57] A. Varga and T. Katayama. Computation of J–inner–outer factorizations of rational matrices. *Internat. J. Robust and Nonlinear Cont.*, 8:245–263, 1998.

[58] D.S. Watkins and L. Elsner. Chasing algorithms for the eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 12:374–384, 1991.

[59] K. Zhou, J.C. Doyle, and K. Glover. *Robust and Optimal Control.* Prentice-Hall, Upper Saddle River, NJ, 1996.