Technical Sciences Academy of Romania
www.jesi.astr.ro

*Journal of Engineering Sciences and Innovation*
*Volume 3, Issue 3 / 2018, pp. 265 - 290*
*http://doi.org/10.56958/jesi.2018.3.3.265*

**E. Electrical and Electronics Engineering**

*Received **10 July 2018*** *Accepted **14 September 2018***
*Received in revised from **3 August 2018***

# Numerical solution of algebraic Riccati equations by Newton's method

**VASILE SIMA**[*]

*National Institute for Research & Development in Informatics, 8–10 Bd. Mareşal Averescu, Bucharest, Romania*

**Abstract.** Improved Newton solvers, with or without line search, for both continuous- and discrete-time algebraic Riccati equations (AREs) are discussed. The basic theory and conceptual algorithm are briefly presented. Algorithmic details, computational steps, and convergence tests are described. The main results of an extensive performance investigation of the Newton solvers are summarized and compared with those obtained with the widely-used MATLAB solvers, care and dare. Randomly generated systems with orders till 2000, as well as the systems from the large COMPl$_e$ib collection of examples, are considered. Significantly improved accuracy, in terms of normalized and relative residuals, and sometimes greater efficiency than for care/dare have been obtained. The results strongly recommend the use of Newton solvers, especially for improving the solutions computed by other solvers.

**Keywords:** Algebraic Riccati equation, numerical linear algebra, numerical methods, optimal control, optimal estimation.

## 1. Introduction

The numerical solution of algebraic Riccati equations (AREs) is a basic algorithmic step in many computational methods for model reduction, filtering, spectral factorization, linear quadratic optimization, $H_2/H_\infty$ robust control, and other domains. Let $Q = Q^T$, $A,\ E \in R^{n\times n}$, $B,\ S \in R^{n\times m}$ and $R = R^T \in R^{m\times m}$, with $E$ nonsingular, and superscript $T$ denoting the transpose. The generalized continuous- and discrete-time AREs (CAREs and DAREs), can be defined by $R(X) = 0$, where

*Correspondence address: vsima@ici.ro

$$R(X) := Q + \mathrm{op}(A)^T X \mathrm{op}(E) + \mathrm{op}(E)^T X \mathrm{op}(A) - \sigma L(X) R^{-1} L(X)^T, \quad (1)$$
$$R(X) := Q + \mathrm{op}(A)^T X \mathrm{op}(A) - \mathrm{op}(E)^T X \mathrm{op}(E) - \sigma \hat{L}(X) \hat{R}(X)^{-1} \hat{L}(X)^T, \quad (2)$$

respectively, where $\sigma = \pm 1$, the operator $\mathrm{op}(M)$, borrowed from numerical linear algebra, represents either $M$ or $M^T$, $R$ or $\hat{R}(X)$, respectively, are assumed nonsingular,

$$L(X) := S + \mathrm{op}(E)^T X B,$$
$$\hat{L}(X) := S + \mathrm{op}(A)^T X B, \qquad \hat{R}(X) := R + \sigma B^T X B, \quad (3)$$

and $X = X^T \in R^{n\mathrm{x}n}$ is unknown. The use of plus sign in front of the last term in (1) and (2) allows to solve more general symmetric matrix equations. For an optimal regulator problem, the operator $\mathrm{op}(M)$ is $\mathrm{op}(M) = M$, while for an optimal estimator problem $\mathrm{op}(M) = M^T$, *input matrix B* is replaced (by duality) by the transpose of the *output matrix* $C \in R^{p\mathrm{x}n}$, and $m$ is replaced by $p$. Often $Q$ and $S$ are given as $C^T \bar{Q} C$ and $S = C^T \bar{S}$, respectively.

The solutions of an ARE are the matrices $X$ for which the *residual* $R(X)$ is zero. Often, a *stabilizing solution*, $X_S$, is sought, so that the matrix pair $(A - \sigma \mathrm{op}(BK(X_s)), E)$ is stable, where

$$K(X) := R^{-1} L(X)^T, \text{ for CARE,} \qquad K(X) := \hat{R}(X)^{-1} \hat{L}(X)^T, \text{ for DARE,} \quad (4)$$

and $\mathrm{op}(K(X_s))$ is the *gain* matrix of the optimal regulator or estimator.

There is an overwhelming literature concerning AREs and their use for solving optimal control and estimation problems; see, e.g., the monographs [1] – [4] for many theoretical results. An integral quadratic performance index in terms of the system state and input of a linear system is used as an optimization criterion for a control problem. The optimal solution, which minimizes this criterion, is expressed as a state-feedback control law and it stabilizes the system. Briefly speaking, this control law achieves a trade-off between the regulation error and the control effort. The optimal estimation or filtering problem, for systems with Gaussian noise disturbances, can be solved as a dual of an optimal control problem, and its solution gives the minimum variance state estimate, based on the system output. The results of an optimal design are often better suited in practice than those found by other approaches. For instance, pole placement (or assignment) may produce large gain matrices, hence high-magnitude control inputs, which might not be acceptable in practice. Solving AREs is also a major computational step in $H_\infty$ robust control theory (e.g., [5]). A recent extended hystorical perspective of scalar and matrix (differential) Riccati equations is given in [6], which also mentions several domains of the control system theory where these equations appear, and includes an extensive bibliography. As a proof of the ubiquity of AREs, it is worth mentioning that in [7], an individualized model predictive control (MPC) for the artificial pancreas has been investigated, where the moving horizon performance index contains a „final" state term defined by the solution of a DARE. Therefore, AREs are now appearing also in applications for the medical domain.

Due to their importance, numerous numerical methods have been proposed for solving AREs; see, for instance, [4,8] and the references therein. There are also several software implementation, e.g., in MATLAB [9], or in the SLICOT Library [10] – [13]. There are both direct and iterative algorithms for solving AREs. The first class includes the (generalized) Schur techniques, e.g., [14] – [17]. These algorithms use a basis of the stable invariant or deflating subspace of a structured, Hamiltonian (for CAREs) or symplectic (for DAREs) matrix (of size $2n$) or matrix pencil (of size $2n$ or $2n+m$). The state-of-the-art MATLAB functions care and dare, and several SLICOT routines implement such algorithms. Relatively recent research, including both theoretical and numerical investigation, has been directed to exploit the Hamiltonian or symplectic structure of the eigenproblem associated to the ARE [18] – [22]. The second class of algorithms has several categories, including matrix sign function techniques, e.g., [23, 24], Newton techniques, e.g., [14, 25], doubling algorithms, e.g., [26,27], or recursive algorithms, e.g., [28]. In particular, [28] addresses CAREs with indefinite quadratic term; the stabilizing solution is found as the limit of solutions of a sequence of CAREs with definite quadratic term.

Newton's method for solving AREs has been investigated by many authors, for instance, [3, 4, 8], [29] – [31]. Moreover, the matrix sign function method, [23, 24, 32, 33], uses a specialized Newton's method to compute the square root of the identity matrix of order $2n$. Newton's method has also been applied in [34] for solving special classes of large-order CAREs, using low rank Cholesky factors of the solutions of the Lyapunov equations built during the iterative rocess [35]. Additional numerical results, for randomly generated systems with $n \leq 600$, and comparison with MATLAB and SLICOT solvers are presented in [36]. However, these specialized solvers require the assumptions that matrix $A$ is structured or sparse and the CARE solution has rank much smaller than $n$. (The possibly sparse structure of $A$, and operations of the form $Ab$ or $A^{-1}b$, with $b \in R^n$, are used.)

Newton's method is attractive as an ARE solver due to several reasons. A main reason is its quadratic convergence in the neighbourhood of an ARE solution. Moreover, with a stabilizing matrix $X_0$, all Newton iterates and their limit are stabilizing. In addition, the computational effort per Newton iteration (for dense matrices) is expressed as a cubic polynomial in $n$ and $m$, while for the best direct algorithms the effort is defined by a cubic polynomial in $2n$ (or $2n+m$) with significantly larger coefficients. Therefore, if convergence is obtained, say, in less than 10 iterations, Newton algorithms are competitive with the direct ones. Another main reason is the improved accuracy which can be obtained.

One drawback of the Newton's method is its dependence on the initial matrix used, $X_0$. This matrix should be stabilizing, i.e., $(A - \sigma \mathrm{op}(BK(X_0)), E)$ should be stable, in order to compute the stabilizing solution $X_S$. Finding a suitable $X_0$ can be a difficult task. Stabilizing algorithms have been proposed, mainly for standard systems (with $E = I_n$, where $I_n$ is the identity matrix of order *n*), e.g., in [25, 31, 37, 38]. However, often these algorithms produce a matrix $X_0$ with a very large norm. Consequently, the following iterates, $X_k$, $k = 1, 2, \ldots$, computed by the Newton's method, could also be large, and the iterative process might need many iterations before convergence, or encounter numerical difficulties. For this reason, Newton's method is best used for iterative improvement of a solution returned by other solver, or as defect correction method [39], delivering the maximal possible accuracy when starting from a good approximate solution. Moreover, it is preferred in implementing certain fault-tolerant systems, which require controller updating, see, e.g., [40] and the references therein.

This paper describes a general solver for CAREs and DAREs, developed by the author (based on [29, 30]), which can solve moderately large dense problems (e.g., $n \le 1000$). The basic theory, conceptual algorithm, and the main implementation details are summarized. The solver has extended functionality and good flexibility, reliability, and efficiency. Its performance has been proven by the numerical results obtained on randomly generated systems and on systems from the COMPl$_e$ib collection [41]. Preliminary results have been reported in [42]. Some results for CAREs are included in [43] – [46] and for DAREs in [44, 47, 48]. How computations can be organized in an iteration for improving efficiency was investigated in [49].

The paper compares the performance of the Newton solvers with or without line search (briefly called as *modified* and *standard* Newton solvers, respectively) with the performance of the state-of-the-art commercial solvers care and dare from MATLAB Control System Toolbox. The MATLAB solvers use a (generalized) eigenvalue approach, based on the results in, e.g., [14, 15, 17].

The organization of the paper is as follows. Section 2 starts by summarizing the basic theory and Newton algorithm for CAREs and DAREs. Section 3 discusses, in separate subsections, some algorithmic and implementation details: computation of the Newton step size, convergence tests, iterative process, etc. Section 4 presents the main results of an extensive performance investigation of the solvers based on Newton's method, in comparison with the MATLAB solvers care and dare. Randomly generated systems with order till 2000, as well as systems from the COMPl$_e$ib collection [41], are considered. Section 5 summarizes the conclusions.

## 2. Basic theory for Newton-based ARE solvers

The algorithm discussed in the sequel is a modification of Newton's method, employing a *line search* procedure to minimize, for CAREs, and reduce, for

DAREs, the residual along the Newton direction. The conceptual algorithm is stated as follows:

**Algorithm N: Newton's method with line search for an ARE**

*Input:* The matrices *E*, *A*, *B*, *Q*, *R*, and *S*, and an initial matrix $X_0 = X_0^T$.

*Output:* The approximate solution $X_k$ of CARE for (1) or DARE for (2).

```
FOR $k = 0, 1, \ldots, k_{\max}$□ , DO
    1. Compute $R(X_k)$. If convergence or non-convergence is detected, return $X_k$
       and/or a warning or error indicator value.
    2. Compute $K_k \coloneqq K(X_k)$ with (4), and $[\mathbf{op}(A]_k)$, where
       $A_k = \mathbf{op}(A) - \sigma B K_k$.
    3. Solve in $N_k$ the continuous- or discrete-time generalized Lyapunov
       equation, (5) or (6), for CARE or DARE, respectively,
           $\mathbf{op}(A_k)^T N_k \mathbf{op}(E) + \mathbf{op}(E)^T N_k \mathbf{op}(A_k) = -R(X_k)$,    (5)
           $\mathbf{op}(A_k)^T N_k \mathbf{op}(A_k) - \mathbf{op}(E)^T N_k \mathbf{op}(E) = -R(X_k)$.    (6)
    4. Find a step size $t_k$ which minimizes (with respect to $t$), for CARE, or
       reduces, for DARE, the squared Frobenius norm of the next residual,
       $\|R(X_k + tN_k)\|_F^2$.
    5. Update $X_{k+1} = X_k + t_k N_k$.
END
```

The usual, „standard” Lyapunov equations have $E = I_n$. Equation (6) is also called *generalized Stein equation*. Note that the SLICOT Lyapunov solvers used can directly work with $[\mathbf{op}(A]_k)$. Since $[\mathbf{op}(A]_k) = A - \sigma\mathbf{op}(BK_k)$, matrix *A* is not actually transposed even if $\mathbf{op}(M) = M^T$, while $\mathbf{op}(BK_k)$ is similarly obtained with no transposition by using a suitable call to the matrix multiplication subroutine DGEMM from Basic Linear Algebra Subprograms (BLAS) [50]. (Explicit matrix transpositions should be avoided in computations.)

Standard Newton algorithm is obtained by taking $t_k = 1$ in Step 4 at each iteration. When the initial matrix $X_0$ is far from a Riccati equation solution, the modified Newton's method, with line search, often outperforms the standard Newton's method.

In theory, the following assumptions are needed.

**Assumptions A:**

1. Matrix *E* is nonsingular.
2. Matrix pair $(\mathbf{op}(E)^{-1}\mathbf{op}(A), \mathbf{op}(E)^{-1}B)$ is stabilizable.
3. Matrix *R* is positive definite $(R > 0)$ for CAREs and nonnegative definite $(R \geq 0)$ for DAREs.
4. A stabilizing solution $X_S$ exists and it is unique.

Note that Assumption 1 is not actually used by the developed solver, contrary to some other solvers (including MATLAB functions care and dare).

The basic properties for the standard and modified Newton algorithms for AREs can then be stated as follows [29]:

**Theorem 1 (Convergence of Algorithm N, standard case)** *If the Assumptions A hold, and $X_0$ is stabilizing, then the iterates of the Algorithm N with $t_k = 1$ satisfy*

(a) *All matrices $X_k$ are stabilizing.*

(b) $X_s \le \cdots \le X_{k+1} \le X_k \le \cdots \le X_1$ .

(c) $\lim_{k\to\infty} X_k = X_s$ .

(d) *Global quadratic convergence: There is a constant $\gamma > 0$ such that*

$$\|X_{k+1} - X_s\| \le \gamma \|X_k - X_s\|^2 , \quad k \ge 1. \tag{7}$$

Note that (7) does not hold for $k = 0$, involving the iterates $X_0$ and $X_1$.

**Theorem 2 (Convergence of Algorithm N for CAREs)** *If the Assumptions A hold, $X_0$ is stabilizing, $(op(E)^{-1}op(A), op(E)^{-1}B)$ is controllable and $t_k \ge t_L > 0$, for all $k \ge 0$, then the iterates of the Algorithm N for CARE satisfy*

(a) *All iterates $X_k$ are stabilizing.*

(b) $\|R(X_{k+1})\|_F \le \|R(X_k)\|_F$ *and equality holds if and only if* $R(X_k) = 0$ .

(c) $\lim_{k\to\infty} R(X_k) = 0$ .

(d) $\lim_{k\to\infty} X_k = X_s$ .

(e) *In a neighbourhood of $X_S$, the convergence is quadratic.*

(f) $\lim_{k\to\infty} t_k = 1$ .

Theorem 2 does not ensure monotonic convergence of the iterates $X_k$ in terms of definiteness, contrary to the standard case (Theorem 1, item (b)). On the other hand, under the specified conditions, Theorem 2 states the monotonic convergence of the residuals to zero, which is not true for the standard algorithm. Numerical experiments support the conjecture that Theorem 2 also holds under the weaker assumption of stabilizability instead of controllability.

Weaker results are available for the modified Newton algorithm for DAREs. One such result [29] states that if $X_k$ is stabilizing, then $N_k$ computed by Algorithm N is a descent direction for $\|R(X_k)\|_F^2$ from $X_k$, unless $X_k = X_S$.

## 3. Algorithmic and implementation issues

Algorithm N and the implemented solver deals with generalized AREs without inverting the matrix *E*. This is very important for numerical reasons, since *E* might be ill-conditioned with respect to inversion, so that large perturbations in the data used might be introduced from the beginning of the calculations. The implementation calls routines from SLICOT Library [11, 13, 51] (www.slicot.org), as well as from LAPACK [52] (www.netlib.org/lapack/) and BLAS, see [50] and the references therein (www.netlib.org/blas/). Standard AREs are solved with the maximal possible efficiency, using suitable customization in the called routines. (Even if a request to solve a generalized ARE is made, the solver checks out if $E$ happens to be identity and a standard ARE is efficiently solved if $E = I_n$). Moreover, both control and filter AREs can be solved by the same solver, using an option („mode") parameter, which specifies the op operator. The matrices $A$ and $E$ are not transposed. It possible to also avoid transposing $C$, for the filter equation, but this is less important and more difficult to implement at the SLICOT Library level, since some existing lower-level routines do not directly cover the transposed case. But this issue was solved at the upper level, in the executable function. Symmetry is used whenever possible. Common subexpressions of matrix products are evaluated only once, and the sequence of multiplications is optimized, depending on the $n$ and $m$ values. A new block algorithm is used for computing the matrix product $MN$, when the result is symmetric (e.g., when $M = \mathbf{op}(E)^T X$ , and $N = GX\mathbf{op}(E)$).

The essential computational procedures involved in Algorithm N will be detailed below.

### 3.1. Removing *S* matrix

Any CARE (1), but also any DARE (2) with nonsingular $R$, can be rewritten in a simpler form, which is more convenient for Algorithm N. Specifically, setting

$$\tilde{A} = A - \sigma \mathbf{op}(BR^{-1}S^T), \qquad \tilde{Q} = Q - \sigma S R^{-1} S^T, \tag{8}$$

after redefining $A$ and $Q$ as $\tilde{A}$ and $\tilde{Q}$, respectively, then (1) and (2) reduce to

$$R(X) = Q + \mathbf{op}(A)^T X \mathbf{op}(E) + \mathbf{op}(E)^T X \mathbf{op}(A) - \sigma \mathbf{op}(E)^T X G X \mathbf{op}(E), \tag{9}$$

$$R(X) = Q + \mathbf{op}(A)^T X \mathbf{op}(A) - \mathbf{op}(E)^T X \mathbf{op}(E) - \sigma \mathbf{op}(A)^T X \hat{G}(X) X \mathbf{op}(A), \tag{10}$$

respectively, where

$$G \coloneqq BR^{-1}B^T, \quad \hat{G}(X) \coloneqq B\hat{R}(X)^{-1}B^T. \tag{11}$$

Simpler forms are obtained in the standard case $(E = I_n)$. The tilde transformations in (8) eliminate the matrix $S$ from the formulas to be used by Newton's algorithms. It is more economical to solve AREs for (9) or (10) than for (1) or (2), respectively, since otherwise the calculations involving $S$ must be performed at each iteration. In this case, the matrix $K_k$ is no longer computed in Step 2, and $A_k$ is given by

$$A_k = \mathbf{op}(A) - \sigma G X_k \mathbf{op}(E), \quad \text{for CARE},$$
$$A_k = \mathbf{op}(A) - \sigma \hat{G}_k X_k \mathbf{op}(A), \quad \hat{G}_k \coloneqq \hat{G}(X_k), \quad \text{for DARE}. \tag{12}$$

Since $R$ was assumed positive definite, the Cholesky factor $R_c$ of $R$ (i.e., $R = R_c^T R_c$ , with $R_c$ upper triangular), can be used to obtain $\tilde{A}$ and $\tilde{Q}$ in (8). Defining $\tilde{B} = BR_c^{-1}$ and $\tilde{S} = SR_c^{-1}$ , the relations (8) are equivalent to

$$\tilde{A} = A - \sigma \mathbf{op}(\tilde{B}\tilde{S}^T), \qquad \tilde{Q} = Q - \sigma \tilde{S}\tilde{S}^T, \tag{13}$$

so just two triangular sets of systems with the same coefficient matrix, $R_c$, should be solved, and two matrix products should be computed for obtaining $\tilde{A}$ and $\tilde{Q}$, after factoring $R$. This is done before starting the iteration loop in Algorithm N. Symmetry is exploited for getting $\tilde{Q}$ via a BLAS [50] symm operation.

When $R$ is not positive definite, then either $UDU^T$ or $LDL^T$ factorization [53] of $R$ can be employed for computing $\tilde{A}$ and $\tilde{Q}$. Similarly, $UDU^T/LDL^T$ factorization of $\hat{R}(X_k)$ can be used for obtaining $\hat{G}_k$, when $\hat{R}(X_k)$ is indefinite. This may happen for DAREs during the iterations of Algorithm N, even if $R > 0$.

### 3.2. Using factored $G$ or $\hat{G}_k$ matrices

When $m$ is smaller enough than $n$ ($m \le cn$, where $c = \frac{3}{4}$, for a CARE, or $c = \frac{3}{5}$, for a standard CARE), a factorization of $G$ can be used instead of $G$ during the iterative process. Specifically, with the notation above, $G = \tilde{B}\tilde{B}^T$, and so, $A_k = \mathbf{op}(A) - \sigma\tilde{B}\tilde{B}^T X_k \mathbf{op}(E)$.

Similarly, if $\hat{R}(X_k) > \mathbf{0}$ , then the Cholesky factor of $\hat{R}(X_k)$, $\hat{R}_c(X_k)$, and a factorization of $\hat{G}_k$ can be used for DARE, if $m \le cn$, with $c = \frac{1}{4}$. Defining $\tilde{B}_k := \tilde{B}(X_k) := B\hat{R}_c(X_k)^{-1}$, then $\hat{G}_k = \tilde{B}_k\tilde{B}_k^T$, $A_k = \mathbf{op}(A) - \sigma\tilde{B}_k\tilde{B}_k^T X_k \mathbf{op}(A)$.

If $\hat{G}_k$ is to be preferred (since $m > cn$), but the norm of $\hat{G}_0$ is too large, then, if possible, the factor $\tilde{B}_k$ is used in the iterative process instead of $\hat{G}_k$, in order to potentially improve the numerical behavior, even if the efficiency somewhat diminishes.

### 3.3. Using $S$ matrix

When $S \ne 0$, but $R$ is ill-conditioned with respect to inversion, the use of formulas (8) will potentially introduce large errors from the beginning of Algorithm N, which will be propagated over the entire iterative process, possibly resulting in slower convergence, and/or an inaccurate computed solution. Using $S$ during the iterations could avoid such degradation. Therefore, an option of the solver allows to avoid the transformations (8), and involve $S$ in all subsequent calculations. In this case, other formulas are needed, since $G$ or $\hat{G}_k$ cannot be used. Specifically, define

$$H_k = \mathbf{op}(E)^T X_k B + S, \qquad F_k := H_k R_c^{-1}, \text{ for CARE,} \qquad (14)$$

$$H_k = \mathbf{op}(A)^T X_k B + S, \qquad F_k := H_k \hat{R}_c(X_k)^{-1}, \text{ for DARE,} (15)$$

with $R_c$ and $\hat{R}_c(X_k)$ introduced above; it is assumed here that $\hat{R}_c(X_k) > 0$ for computing $F_k$ in (15). ($(H_k$ is a convenient notation for $L(X_k))$ Then, the residual $R(X_k)$ and the matrix $\mathrm{op}(A_k)$ can be computed using

$$R(X_k) = Q + \mathbf{op}(A)^T X_k \mathbf{op}(E) + \mathbf{op}(E)^T X_k \mathbf{op}(A) - \sigma F_k F_k^T, \qquad (16)$$

$$\mathbf{op}(A_k) = A - \sigma \mathbf{op}(\tilde{B} F_k^T), \quad \text{for CARE,} \qquad (17)$$

$$R(X_k) = Q + \mathbf{op}(A)^T X_k \mathbf{op}(A) - \mathbf{op}(E)^T X_k \mathbf{op}(E) - \sigma F_k F_k^T, \qquad (18)$$

$$\mathbf{op}(A_k) = A - \sigma \mathbf{op}(\tilde{B}_k^T F_k^T), \quad \text{for DARE,} \qquad (19)$$

where $\widetilde{B}$ and $\widetilde{B}_k$ have been defined above.

If, however, $R$ or $\hat{R}(X_k)$ is (numerically) indefinite, then the needed formulas follow directly from (2) – (4), namely,

$$R(X_k) = Q + \mathbf{op}(A)^T X_k \mathbf{op}(E) + \mathbf{op}(E)^T X_k \mathbf{op}(A) - \sigma H_k K_k, \qquad (20)$$

$$R(X_k) = Q + \mathbf{op}(A)^T X_k \mathbf{op}(A) - \mathbf{op}(E)^T X_k \mathbf{op}(E) - \sigma H_k K_k, \qquad (21)$$

$$\mathbf{op}(A_k) = A - \sigma \mathbf{op}(B K_k), \qquad (22)$$

involving $UDU^T$ or $LDL^T$ factorization of $R$ or $\hat{R}(X_k)$. Moreover, symmetry of the matrix product $H_k K_k$ is taken into account, since the solver computes either the upper or lower triangle of $R(X_k)$.

The implementation is optimized by using common subexpressions when computing $R(X_k)$ and $\mathrm{op}(A_k)$, taking also into account the ratio between $n$ and $m$. The formulas used and their proofs are given in [49].

## 3.4. Initialization and main options

The iteration is started by an initial (stabilizing) matrix $X_0$, which may not be given on input, if the zero matrix can be used. If $X_0$ is not stabilizing, and finding $X_S$ is not required, Algorithm N could converge to another ARE solution.

Since the solution computed by a Newton algorithm generally depends on initialization, another option specifies if the stabilizing solution $X_S$ is to be found. This is assumed to be the case in the sequel. The initial matrix $X_0$ must then be

stabilizing, and a warning is issued if this property does not hold; moreover, if the computed $X$ is not stabilizing, an error is issued.
An option specifies whether to use standard Newton's method, or one of the modified Newton's method variations, discussed in a paragraph below, which employ a line search procedure.
Another option is to scale the matrices $A_k$ and $E$ (if $E$ is general) for solving Lyapunov equations, and suitably update their solutions. Note that the LAPACK (www.netlib.org/lapack/) subroutines DGEES and DGGES [52], which are called by the SLICOT standard and generalized Lyapunov solvers, respectively, to compute the real Schur(-triangular) form, do not scale the coefficient matrices. Just column and row permutations are performed, to separate isolated eigenvalues. For some examples from the DARE benchmark collection [54], and no scaling, this fact created troubles: the convergence was not achieved in a reasonable number of iterations. This difficulty was removed by the scaling included in the Newton solver. Moreover, scaling allows sometimes to compute more accurate solutions and/or use less iterations than in the case with no scaling.
For CAREs, either the matrices $B$ and $R$ (or its Cholesky factor, $R_c$)), or the matrix $G = G_T$ may be given. For DAREs, it is compulsory to provide $B$ and $R$ (not $R_c$).
A maximum allowed number of iteration steps, $k_{\max}$, is specified on input, and the number of iteration steps performed, $S$, is returned on exit.

## 3.5. Computing the step size

The optimal step size $t_k$ is given by

$$t_k = \arg\min_t \|R(X_k + tN_k)\|_F^2 . \tag{23}$$

If certain standard conditions hold [29], an optimal $t_k$ exists, and it is in the „canonical" interval [0,2]. Since solving (23) for a DARE is expensive, an approximate value $t_k$ is found numerically as the argument of the minimal value in [0,2] of a polynomial of order 4. For a CARE, the residual in $X_k + tN_k$ can be written as

$$R(X_k + tN_k) = (1 - t)R(X_k) - t^2 V_k \ , \tag{24}$$

where $V_k = \mathbf{op}(E)^T N_k G N_k \mathbf{op}(E)$. Hence, the problem (23), which is equivalent to the minimization of $\|R(X_k + tN_k)\|_F$ , is replaced by the minimization of the quartic polynomial [29]

$$f_k(t) = \mathbf{trace}\left(R(X_k + tN_k)^2\right) = \alpha_k(1 - t)^2 - 2\beta_k(1 - t)t^2 + \gamma_k t^4 \ ,\tag{25}$$

where

$$\alpha_k = [\mathbf{trace}(R(X_k)]^2), \qquad \beta_k = \mathbf{trace}(R(X_k)V_k), \qquad \gamma_k = \mathbf{trace}(V_k^2) \ .$$

For a DARE, the same polynomial (25) is used, but it can only approximate the value of $\mathbf{trace}(R(X_k + tN_k)^2)$, since the underlying function is rational, not polynomial. The approximation is obtained by replacing the function denominator by the second order Taylor series approximant at $t = 0$, therefore i can be useful when $t$ is small enough. For instance, if $t < \frac{1}{\|\hat{G}_k N_k\|}$, where $\|\cdot\|$ is any submultiplicative norm, then $\hat{R}(X_{k+1}) := R + B^T(X_k + t_k N_k)B$ is nonsingular, if $\hat{R}(X_k)$ is nonsingular. Since $t_k$ is chosen from the interval [0,2], the condition above is satisfied if $\|\hat{G}_k N_k\| < \frac{1}{2}$. It can be shown [29] that if $X_k$ is stabilizing, then either $N_k$ is a descent direction for $\|R(X_k)\|_F^2$ , or $X_k = X_s$. But the stabilizing property is not guaranteed, at least for $t \in [0,2]$. When $\|\hat{G}_k N_k\|$ is large (usually, at the beginning of the iterative Newton process), the acceptable step sizes $t_k$ could be too small, and the progress of the iteration could be too slow.

The coefficients of the polynomial $f_k(t)$ for DARE are computed using the same relations as above, but $V_k$ is given by $V_k = \mathbf{op}(A_k)^T N_k \hat{G}_k N_k \mathbf{op}(A_k)$. Therefore, the computed $t_k$ may only approximately solve (23). To find $t_k$ for both CAREs or DAREs, a cubic polynomial (the derivative of $f_k(t)$) is set up, whose real roots in [0,2], if any, are candidates for the solution of the (approximate) minimum residual problem. The roots of this cubic polynomial are computed by solving an equivalent 4-by-4 standard or generalized eigenproblem, following [55]. Either the QR or the QZ algorithm [53] is chosen, depending on the magnitude of the polynomial coefficients. A candidate solution $t_k$ should be real, placed in the interval [0,2], and satisfying $f_k''(t_k) \geq 0$ , where $f_k''$ denotes the second derivative of $f_k$. If no solution is found, then $t_k$ is set equal to 1. If two solutions are found, then $t_k$ is set to the value corresponding to the minimum residual.

## 3.6. Iterative process

If $X_0$ is given, the algorithm computes the initial residual $[R(X]_0)$ and the matrix $\mathbf{op}(A_0)$, using some of the formulas (8) – (10), (12), or (14) – (22), as needed, for $k = 0$ . If no initial matrix $X_0$ is given, i.e., if $X_0 = 0$, then $[R(X]_0) = \tilde{Q}$ and $[\mathbf{op}(A]_0) = \tilde{A}$. At the beginning of the iteration *k*, $0 < k \leq k_{\max}$□, the algorithm decides to terminate or continue the computations, based on the current normalized residual $r_k$ (and possible on relative residual $r_r(X_k)$), defined below. (At $k = 0$, the calculations continue, so that even a good initialization can be improved.)

The basic stopping criterion for the iterative process is defined in terms of a *normalized residual*, $r_k \coloneqq r(X_k)$, and a tolerance τ. If

$$r_k \coloneqq \| R(X_k) \|_F / \max(1, \|X_k\|_F) \le \tau \quad , \qquad (26)$$

the iterative process is successfully terminated at iteration *k*, and $X_k$ is the computed approximate solution. If $\tau \le 0$, a default tolerance is used, computed by one of the formulas below for CARE and DARE, respectively,

$$\tau = \min\left\{\varepsilon_M \sqrt{n(\|E\|_F(2\|A\|_F + \|\tilde{B}\|_F^2\|E\|_F) + \|Q\|_F)}, \frac{\sqrt{\varepsilon_M}}{10^3}\right\},$$

$$\tau = \min\left\{\varepsilon_M \sqrt{n(\|A\|_F^2(1 + \|\tilde{B}_0\|_F^2) + \|E\|_F^2 + \|Q\|_F)}, \frac{\sqrt{\varepsilon_M}}{10^3}\right\},$$

where $\varepsilon_M$ is the relative machine precision. (The factor $\|\tilde{B}\|_F^2$ is replaced by $\|G\|_F$ for CARE, when *G* is given, and $\|\tilde{B}_0\|_F^2$ is replaced by $\|\hat{G}_0\|_F$ if $\hat{R}(X_0)$ is indefinite, for DARE.) The second operand of min in the two formulas above was introduced to prevent deciding convergence too early for systems with very large norms for *A*, *E*, $\widetilde{B}$ (or *G*), or $\widetilde{B}_0$ (or $\widetilde{G}_0$), and/or *Q*.

The termination criterion involving (26) might not be satisfied in a reasonable number of iterations (or never, due to accumulated rounding errors), for systems with very large norms of the matrices *A*, *E*, $\widetilde{B}$ or $\widetilde{B}_0$ (or *G* or $\widetilde{G}_0$), and/or $Q$, and a small norm of the solution *X*. However, an acceptable approximate solution might be much earlier available. Therefore, the MATLAB-style *relative residual*, $r_r(X_k)$, which includes the Frobenius norms of the matrix terms in (1) or (2) in the denominator of its formula, is also tested at iterations $\tilde{k} = 10 + 5q$, $q = 0,1,\ldots$, and it might produce the termination of the iterative process, instead of the criterion based on the normalized residual. This test is not performed at each iteration in order to reduce the additional computation costs, and to increase the chances of termination via the normalized residual test. If min( $r_k$, $r_r(X_{\tilde{k}})$ ) $> \tau$, a standard (if $E = I_n$) or generalized Lyapunov equation (5) or (6) is solved in $N_k$ (the Newton direction), using SLICOT subroutines.

Another test is to check out if updating $X_k$ is meaningful. The updating is done if $t_k \|N_k\|_F > \varepsilon_M \|X_k\|_F$. If this is the case, then $X_{k+1} = X_k + t_k N_k$ is set, and the updated matrices $R(X_{k+1})$ and $\mathrm{op}(A_{k+1})$ are computed. Otherwise, the iterative process is terminated and a warning value is set, since no further significant, but only marginal improvements can be expected, eventually after many additional iterations. Although the computation of the residual $R(X_k + t_k N_k)$ can be efficiently performed by updating the residual $R(X_k)$, the original data is used, since the updating formula (24) could suffer from severe numerical cancellation, and hence its use could compromise the accuracy of the intermediate results. Moreover, if the currently chosen step was not a Newton step, but the residual norm increased compared to the previous iteration, i.e., $\|[R(X]_{k+1})\|_F \geq \|[R(X]_k)\|_F$, but it is less than 1, and the normalized residual is less than $\varepsilon_M^{\frac{1}{4}}$, then the previous iterate is restored, the iterative process is terminated and a warning value is set. Otherwise, the iteration continues.

Sometimes, mainly in the first iterations, the computed optimal steps $t_k$ are too small, and the residual decreases too slowly. This is called *stagnation*, and remedies are used to escape stagnation, as described below. The chosen strategy was to set $t_k = 1$ when stagnation is detected, but also when $t_k < 0.5$, $\varepsilon_M^{\frac{1}{4}} < r_k < 1$, and $\|[R(X]_k + t_k N_k)\|_F \leq \mathbf{10}$, if this happens during the first 10 iterations. The motivation for this strategy is that if the residual is small enough after the first few iterations, the use of a standard Newton step could further reduce the residual faster than a Newton algorithm with small step sizes.

In order to detect stagnation, the last computed $k_B$ residual Frobenius norms are stored in an array RES. If $\|[R(X]_k + t_k N_k)\|_F > \tau_s \left\|[R(X]_{k-k_B})\right\|_F > \mathbf{0}$, then $t_k = 1$ is used instead. The implementation uses $\tau_s = 0.9$ and sets $k_B = 2$, but values as large as 10 can be used by changing this parameter. The first $k_B$ entries of array RES are reset to 0 whenever a standard Newton step is applied.

### 3.7. Line search strategies

Other line search stategies may be chosen besides the *pure line search strategy*, which uses a solution $t_k$ of the (approximate) minimization of the quartic polynomial (25) at each iteration *k*. Specifically, in the *combined strategy*, line search is employed in the beginning of the iterative process, but the algorithm switches to the standard method when the normalized residual is smaller than a specified (or default) tolerance. This strategy is motivated by the remark that when the normalized residual is small enough, line search cannot offer sensible improvements, and standard algorithm converges with a fast rate. Moreover, $t_k$ will be close to 1 in such a case, and typically there will be no sensible difference between the values of $\|[R(X]_{k+1})\|_F$ computed for $t_k$ and for 1. Therefore, the calculations for finding $t_k$ can be avoided.

In the *hybrid strategy*, a standard Newton step is tried first and used if
$\|[R(X]_k + N_k)\|_F < 10^{-3}\|[R(X]_k)\|_F$, **or** $r_k < \mathbf{100}n\varepsilon_M$ **;**
otherwise, the step corresponding to the (approximate) line search procedure is tried, and that step which gives the smallest residual is selected and used at that iteration.

Finally, the *backtracking strategy*, proposed in [29] for DAREs, is a special hybrid strategy in which the selected step is only taken provided there is a sufficient residual norm decrease. Otherwise, the step size is reduced until a sufficient decrease is eventually obtained. If this is not the case, or stagnation is detected, then a standard Newton step is used. This approach can increase the speed of the iterative process.

## 3.8. Memory storage issues

The arrays holding the data matrices $A$ and $E$ are unchanged on exit, except for $A$ when $S \neq 0$, but $S$ should and could be removed from ARE using (8). In this special case, $\widetilde{A}$ is returned. Array Q stores matrix $Q$ on entry and the computed solution $X_S$ on exit. For CARE, array B stores either $B$ or $G$. If $m \leq cn$, with $c$ defined in Subsection 3.2, and the Cholesky factor $R_c$ (for CARE) or $\hat{R}_c(X_s)$ (for DARE) can be computed, then the array B, storing $B$ on input, returns the matrix $\widetilde{B}$ or $\widetilde{B}_S$, respectively. Otherwise, array B is unchanged on exit. Similarly, the array R, storing $R$ on input, may return either the Cholesky factor of $R$ (for CARE) or of $\hat{R}(X_s)$ (for DARE), if it can be computed, or the factors of the $UDU^T$ or $LDL^T$ factorization of that matrix, if it is found to be numerically indefinite. In the last case, the interchanges performed for the $UDU^T$ or $LDL^T$ factorization are stored in an auxiliary integer array. The finally computed normalized residual is also returned. Moreover, approximate closed-loop system poles, as well as $\min(s,50)+1$ values of the residual norms, normalized residuals, and Newton steps are returned in the working array.
Either the upper, or lower triangles, not both, of the symmetric matrices $Q$, $R$, $X_k$, and $G$ or $\hat{G}_k$ if used, need to be stored. (Note that if the lower triangle of $R$ should be used, the Cholesky factorization is $R =: R_c R_c^T$, with $R_c$ lower triangular, but the computations are similar. The same is true for $\hat{R}(X_k)$)
When possible, pairs of symmetric matrices are stored economically, to reduce the workspace requirements, but preserving the two-dimensional array indexing, for efficiency. Specifically, the upper (or lower) triangle of $X_k$ and the lower (upper) triangle of $R(X_k)$ are concatenated along the main diagonals in a two-dimensional $n(n+1)$ array, and similarly for $G$ (or $\hat{G}_k$) and a copy of the matrix $Q$, if $G$ (or $\hat{G}_k$) is used. Array Q itself is also used for temporarily storing the residual $R(X_k)$, as well as the intermediate matrices $X_k$ and the final solution.
The optimal size of the needed real working array can be queried, by setting its length to -1 . Then, the solver returns immediately, with the first entry of that array set to the optimal size, which could be used in the next solver call.

## 4. Numerical results

This section presents some results of an extensive performance investigation of the solvers based on Newton's method. The numerical results have been obtained on an Intel Core i7-3820QM portable computer at 2.7 GHz, with 16 GB RAM, with the relative machine precision $\varepsilon_M \approx 2.22 \times 10^{-16}$, using Windows 7 Professional (Service Pack 1) operating system (64 bit), Intel Visual Fortran Composer XE 2015 and MATLAB 8.6.0.267246 (R2015b). The SLICOT-based MATLAB executable MEX-function has been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

Some results for randomly generated CAREs, from a uniform distribution in the (0,1) interval, are described in [46]. Stable standard and descriptor systems, with $Q$ and $R$ not identity matrices, and nonzero matrix $S$, have been tried. The initial matrix $X_0$ has been zero, and the dimensions $n$ and $m$ have been set as $n = 200:200:1000$, $m = 200:200:n$ (in a MATLAB notation), but also as $n = m = 2000$. Both standard and modified Newton solvers have been significantly more accurate (with one exception for the standard solver, mentioned below), and almost always faster than care. The mean number of iterations has been 5.33 and 5.66 for the two solvers, respectively. Details are given in [46]. Other results, not yet published, are summarized below. Figure 1 and Fig. 2 show the normalized residuals and the CPU times (obtained using the MATLAB functions tic and toc), respectively, when using standard Newton solver and care. The ordinate axes are scaled logarithmically, for better clarity, since the values vary significantly. The large error for an example with $n = 600$, $m = 200$, and $\mathbf{op}(M) = M^T$ is not typical. The modified Newton solver performed better, see [46].

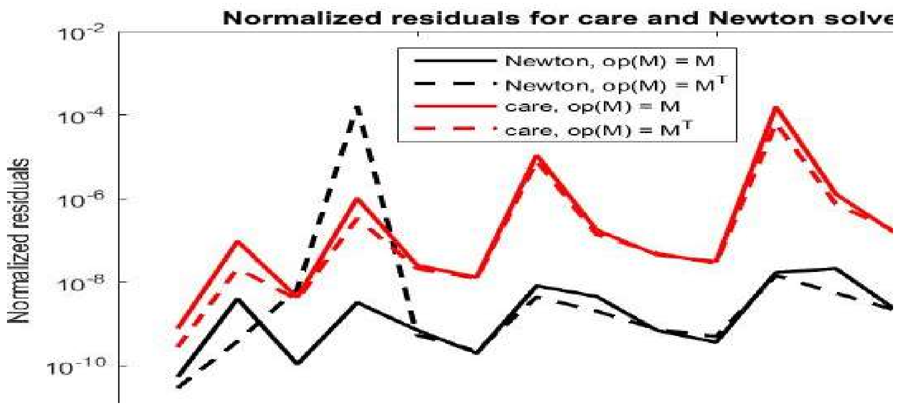

Fig. 1. Normalized residuals for random examples with general matrix $E$ using MATLAB function care and standard Newton solver; $n = 200:200:1000$, $m = 200:200:n$.

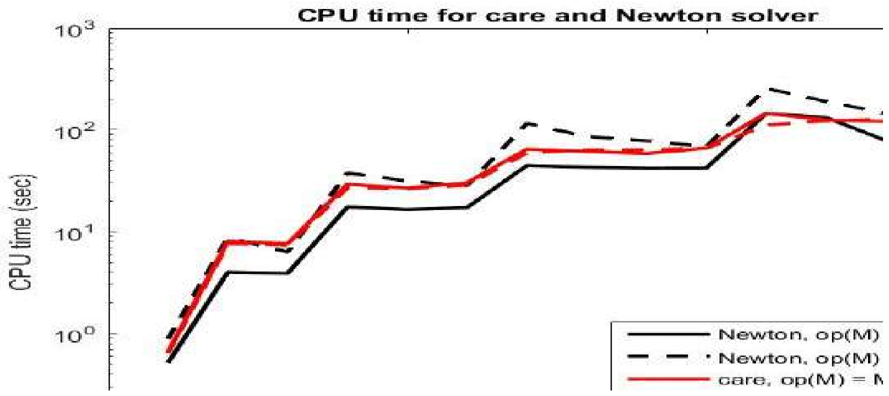

Fig. 2. CPU times for random examples with general matrix $E$ using MATLAB function care and standard Newton solver; $n = 200 : 200 : 1000$ , $m = 200 : 200 : n$ .

Other tests used the linear systems from the COMPl$_e$ib collection [41], which contains 124 standard continuous-time examples, with several variations, giving a total of 168 problems. All but 16 problems (for systems with $n > 2000$ , with matrices in sparse format) have been tried. For testing purposes, these examples have also been considered as being of discrete-time type. The matrices $Q$, $R$, and $S$ have been set to $Q = I_n$ , $R = I_m$ , and $S = 0$ .

A set of tests used $X_0$ for systems with $A$ stable (in a continuous- or discrete-time sense); otherwise, the algorithm in [25], for CAREs, and in [56], for DAREs, has been tried, and when it failed to deliver a stabilizing initialization, the solution provided by the MATLAB function care or dare, respectively, has been used. The function care failed to solve the CARE for example REA4, which is not stabilizable. Also, dare could not solve 63 problems, which did not satisfy the needed conditions for the existence of a finite stabilizing solution. These examples have been removed from the tests. In addition, other five DARE examples, namely WEC1, WEC2, WEC3, HF2D_CD4, and HF2D_CD6 have been excluded. For these examples, the solution computed by dare had a very large Frobenius norm (of order $10^{13}$ for WEC examples, $10^{10}$ and $10^{11}$ for the two HF2D examples), and relatively large normalized residuals (of order $10^{-4}$ or larger for WEC1 – WEC3, $10^{-7}$ and $10^{-6}$, for the other two examples). Such matrices proved to offer a poor initialization for Newton's method. A zero initialization was used for 44 CAREs and 7 DAREs with stable examples. Stabilization algorithm was tried on 107 CAREs and 82 DAREs with unstable systems, and succeeded for 91 and 55 examples, respectively. Failures occurred for 16 CARE and 27 DARE examples. With default tolerance, modified Newton solver improved the accuracy of the care solution for 15 out of these 16 examples.

Other tests, with $X_0$ computed by the stabilization algorithm also for stable systems, or with $X_0$ returned by MATLAB functions for all examples, have also been performed. The last set of tests shows the performance of the Newton solver in refining a solution computed by another solver. Most of the results shown in the figures below have been obtained without balancing the matrices of Lyapunov equations.

For CAREs, and initialization by 0 or by the algorithm in [25], if possible, or by care, otherwise, the modified solver needed more iterations than the standard solver for 10 examples only. However, the mean number of iterations was about 11, for the modified solver, and 15.2, for the standard solver. Figure 3 shows the number of iterations of the Newton solver with line search.

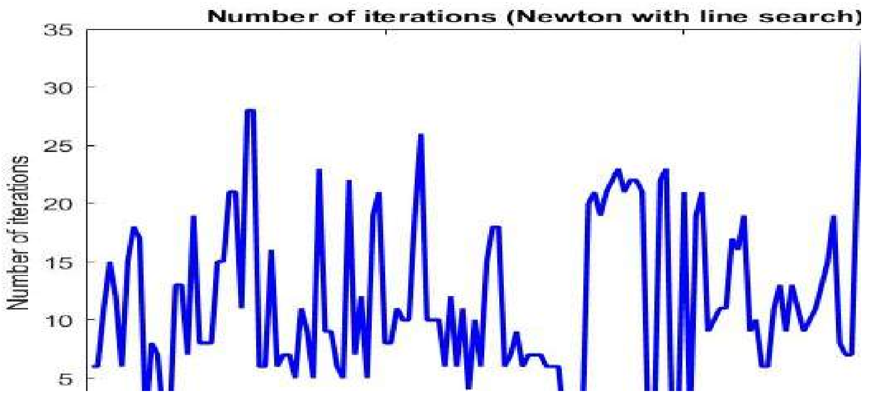

Fig. 3. Number of iterations performed by the Newton solver with line search for examples from the COMPl$_e$ib collection; initialization by 0, algorithm in [25] or care.

Figure 4 shows the normalized residuals for the COMPl$_e$ib examples using care and standard Newton solver with default tolerance. For the TL example (numbered as 61 in the figure), the normalized residual is $2.13 \cdot 10^3$ when using care. The matrices $A$ and $B$ of this example have norms of order $10^{14}$ and are poorly scaled (the minimum magnitude in $A$ is of order $10^{-4}$). By the eigenvalue test, $A$ was taken as stable, but standard Newton solver reported a singular Lyapunov equation. However, the modified Newton solver succeeded to solve the CARE in 12 iterations, with a normalized residual of $1.09 \cdot 10^{-3}$ (see [46]), and its results are used for the TL example in the next figures.

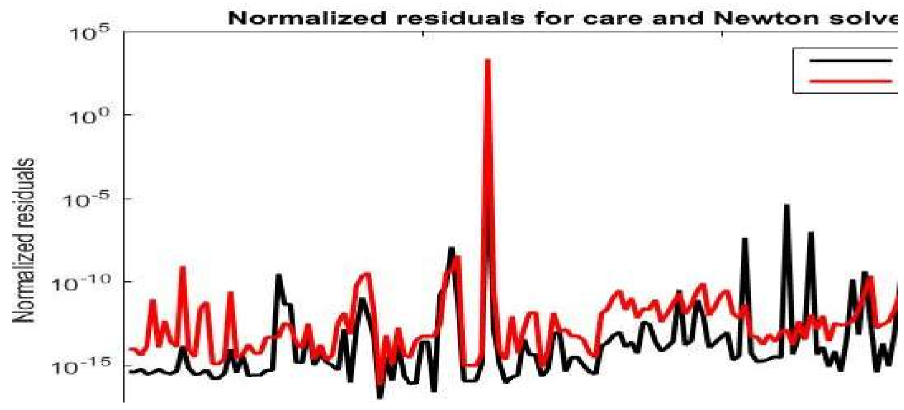

Fig. 4. Normalized residuals for examples from the COMPl$_e$ib collection, using MATLAB function care and standard Newton solver, default tolerance, and initialization by 0, stabilization algorithm in [25] or care.

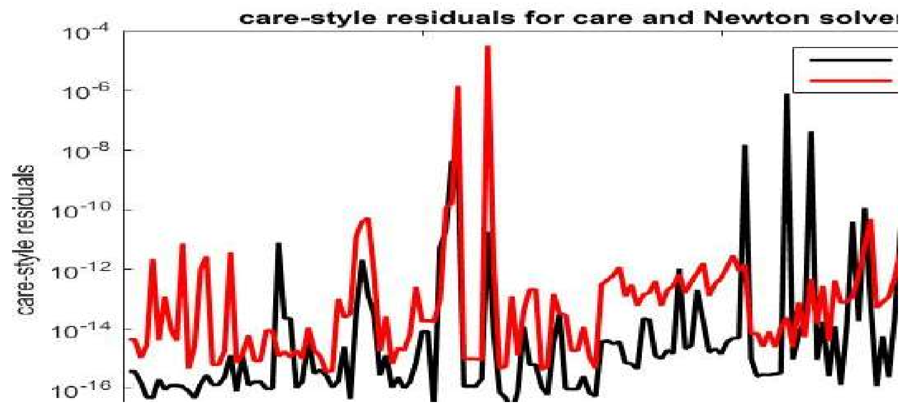

Fig. 5. Relative residuals for examples from the COMPl$_e$ib collection, using MATLAB function care and standard Newton solver, default tolerance, and initialization by 0, stabilization algorithm in [25] or care.

Figure 5 shows the relative residuals, computed in a similar manner with that used in care. The maximum value is $8.98 \cdot 10^{-9}$ for the modified Newton solver (for example ROC5), $3.16 \cdot 10^{-5}$ for care (for TL), and 1 for the standard Newton solver (for TL). Omitting TL, the last two values changed to $10^{-6}$ and $10^{-7}$, respectively. Figure 6 shows the relative residuals for care and both Newton solvers.

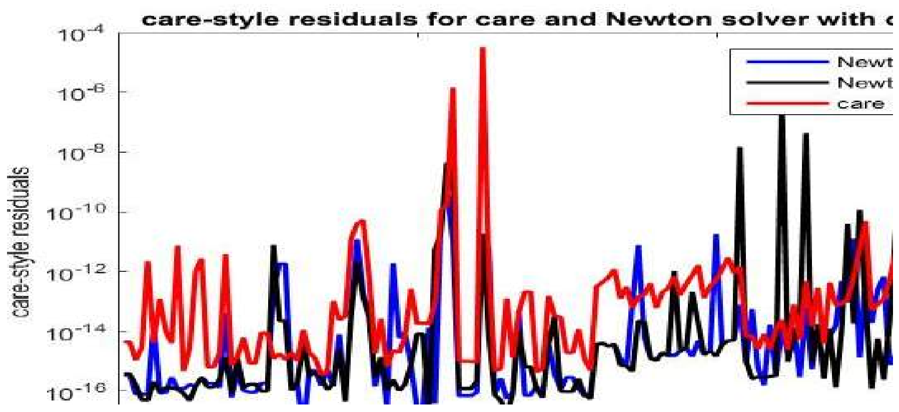

Fig. 6. Relative residuals for examples from the COMPl$_e$ib collection, using MATLAB function care and Newton solvers with or without line search, default tolerance, and initialization by 0, stabilization algorithm in [25] or care.

Similarly, Fig. 7 shows the elapsed CPU times for care and standard Newton solver. This solver was globally over 25% slower than the solver with line search, and over 200% slower than care. The main reason is that, with the chosen initialization, some large examples (mainly, 15 examples in the HF2D class, with numbers between 80 and 103) required at least 27 iterations (and at least 19 iterations, for the modified solver). With initialization provided by care, one iteration was needed for all examples, but TL. Clearly, a good initialization could significantly reduce the number of iterations.

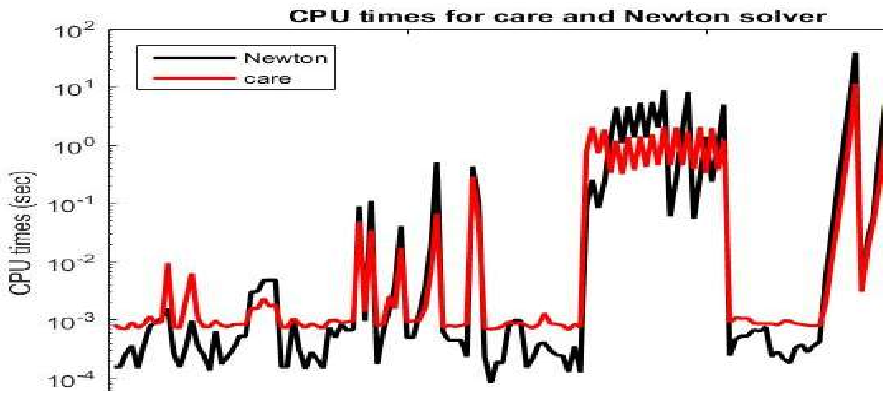

Fig. 7. Elapsed CPU time for examples from the COMPl$_e$ib collection, using MATLAB function care and standard Newton solver, default tolerance, and initialization by 0, stabilization algorithm in [25] or care.

With the chosen initialization, standard Newton solver succeeded to obtain smaller relative residuals than care for 131 examples, out of 150.

The bar graph from Fig. 8 shows the accuracy improvement. The height of the $i$ -th vertical bar indicates the number of examples for which the improvement was between $i-1$ and $i$ orders of magnitude. The number of examples in the six bins with nonzero values are 63, 44, 32, 8, 2, and 1, corresponding to improvements till one order of magnitude, between one and two orders of magnitude, and so on. Note that the first bin includes 19 examples for which care produced smaller relative residuals. With care initialization, both Newton solvers improved care relative residuals for all 150 examples, see [46].

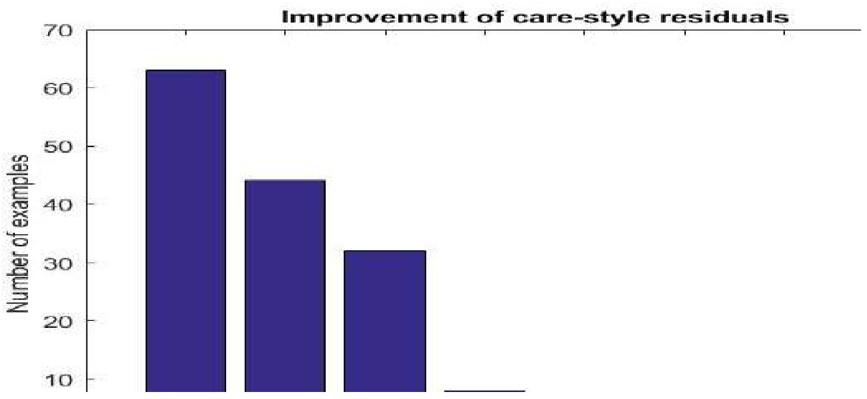

Fig. 8. Bar graph showing the improvement of relative residuals for examples from the COMPl$_e$ib collection, using standard Newton solver, default tolerance, and initialization by 0, stabilization algorithm in [25] or care. The height of the $i$ -th vertical bar indicates the number of examples for which the improvement was between $i-1$ and $i$ orders of magnitude.

Similar results are obtained for DAREs. Figure 9 displays the normalized residuals for examples from the COMPl$_e$ib collection, considered as discrete-time systems, using MATLAB function dare and both Newton solvers, with default tolerance and dare initialization. With few exceptions, Newton solver is either comparable with dare or it improved the normalized residuals, sometimes with several orders of magnitude. For three examples (TMD, ROC5, and ROC7, numbered as 71, 78, and 80, respectively, in Fig. 9), Newton solver obtained normalized residuals of order $10^{-31}$. These values were truncated to $10^{-20}$, in order to improve the resolution. However, for four examples (HF2D_IS7, HF2D_CD5, HF2D17, and HF2D18, numbered as 59, 61, 69, and 70, respectively), standard Newton solver obtained clearly worse results than dare, but modified solver performed better than standard solver for these examples [48]. For other examples, the two Newton solvers got comparable residuals.

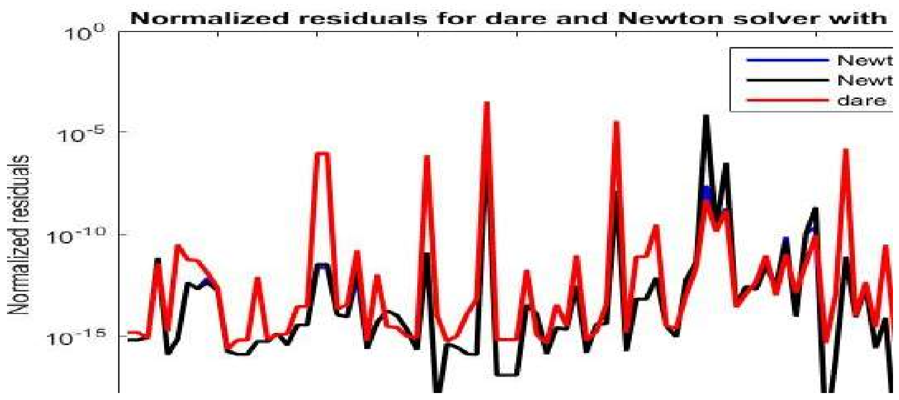

Fig. 9. Normalized residuals for examples from the COMPl$_e$ib collection (taken as discrete-time systems), using MATLAB function dare and Newton solvers with and without line search, with default tolerance and dare initialization.

Similarly, Fig. 10 plots the MATLAB-style relative residuals. The two Newton options are again comparable, except for five examples (HF2D_IS7, HF2D_CD5, HF2D15, HF2D17, and HF2D18, numbered as 59, 61, 67, 69, and 70, respectively). For the last two examples, standard method gave smaller residuals than line search method.

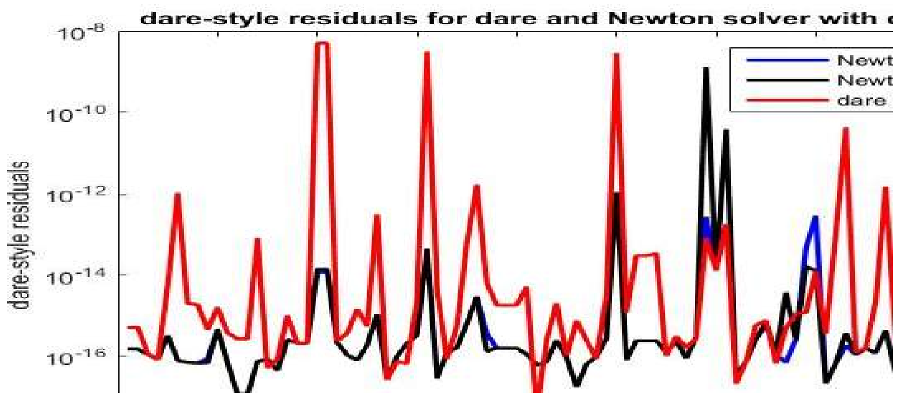

Fig. 10. MATLAB-style residuals for examples from the COMPl$_e$ib collection, using MATLAB function dare and Newton solver with and without line search, default tolerance and dare initialization.

Figure 11 shows the corresponding elapsed CPU times for dare and Newton solver with or without line search, and with or without balancing of the Lyapunov equations matrices. Balancing may slightly increase the CPU time, but not always. For most examples, standard Newton solver is the fastest, and dare is slower by about one order of magnitude. The ratio between the sums of the elapsed CPU times for dare and for the modified Newton solver was about 3.3, while for the other three variants (denoted LS BAL, STD, and STD BAL in Fig. 11) the ratios have values over 6.

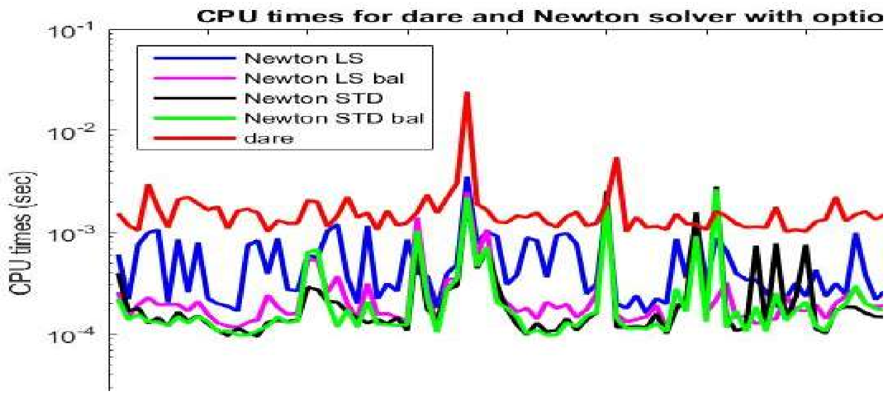

Fig. 11. Elapsed CPU time for examples from the COMPl$_e$ib collection, using MATLAB function dare and Newton solver with options, default tolerance and dare initialization.

Tests have also been performed with initialization $X_0 = 0$ , if $A$ is stable, or computed by the algorithm in [56], if possible, when $A$ is unstable, and by dare, otherwise. Most of the results are comparable with those presented above, except that the number of iterations were bigger for some examples. The DARE corresponding to example CSE1, for which matrix $A$ is stable, could not be solved by the Newton solver with $X_0$ set to zero, since a (numerically) singular Lyapunov equation was encountered during the iterative process. It is worth mentioning that $A$ is highly ill-conditioned, with condition number exceeding $3 \cdot 10^{16}$. But this DARE has been solved with initialization provided by the stabilization algorithm in [56]. The bar graph from Fig. 12 shows the improvement obtained, compared to dare, using Newton solver with line search, default tolerance and initialization chosen as mentioned above. The number of examples in the six bins are 50, 17, 6, 3, 3, and 3, corresponding to improvements till one order of magnitude, between one and two orders of magnitude, and so on.

## 5. Conclusions

Basic theory and improved algorithms for solving continuous- or discrete-time algebraic Riccati equations using Newton's method with or without line search have been presented. Algorithmic details for the developed solvers, the main computational steps and convergence tests are described. The usefulness of such solvers is demonstrated by the results of an extensive performance investigation of their numerical behavior, in comparison with the results obtained calling the widely-used MATLAB functions care and dare. Randomly generated systems with orders till 1000 (and even 2000), as well as the systems from the large COMPl$_e$ib collection, are considered. The numerical results most often show significantly improved accuracy (measured in terms of normalized and relative residuals), and greater efficiency. The results strongly recommend the use of such algorithms, especially for improving, with little additional computing effort, the solutions computed by other solvers.

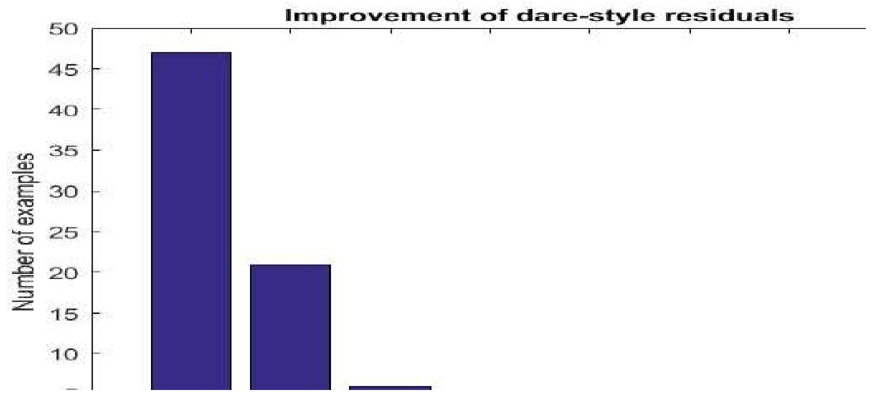

Fig. 12. Bar graph showing the improvement of the MATLAB-style residuals for examples from the COMPl$_e$ib collection, using Newton solver with line search, default tolerance and initialization by $X_0 = 0$ , or by the algorithm in [56], if possible, and by dare, otherwise. The height of the *i*-th vertical bar indicates the number of examples for which the improvement was between $i - 1$ and *i* orders of magnitude.

## Acknowledgments

This work was partially supported by the Institutional research programme PN 1819 „Advanced IT resources to support digital transformation processes in the economy and society – RESINFO-TD" (2018), project PN 1819-01-01, „Modeling, simulation, optimization of complex systems and decision support in new areas of IT&C research", funded by the Ministry of Research and Innovation, Romania.

## References

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

[2] Bini D. A., Iannazzo B., Meini B., *Numerical Solution of Algebraic Riccati Equations*, SIAM, Philadelphia, PA, 2012.

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

[4] Mehrmann V., *The Autonomous Linear Quadratic Control Problem. Theory and Numerical Solution*, Series Lect. Notes in Control and Information Sciences, Thoma M.,Wyner A., Eds., vol. 163, Springer-Verlag, Berlin, 1991.

[5] Francis B. A., *A Course in* $H_\infty$ *Control Theory*, Series Lect. Notes in Control and Information Sciences, vol. 88, Springer-Verlag, New York, 1987.

[6] Jungers M., *Historical perspectives of the Riccati equations*, IFAC-PapersOnLine, vol. 50, no. 1, 2017, 9535–9546, 20th IFAC World Congress, http://www.sciencedirect.com/science/article/pii/S2405896317322176

[7] Messori M., Incremona G. P., Cobelli C., Magni L., *Individualized model predictive control for the artificial pancreas. In silico evaluation of closed-loop glucose control*, IEEE Control Syst. Mag., **38**, no. 1, 2018, p. 86–104.

[8] Sima V., *Algorithms for Linear-Quadratic Optimization*, Pure and Applied Mathematics: A Series of Monographs and Textbooks, Taft E. J., Nashed Z. (Series Eds.), vol. 200, Marcel Dekker, Inc., New York, 1996.

[9] The MathWorks, Inc., *Control System Toolbox User's Guide. Version 9*, 3 Apple Hill Drive, Natick, MA, 01760–2098, 2011.

[10] Benner P., Kressner D., Sima V., Varga A., *Die SLICOT-Toolboxen für Matlab*, at — Automatisierungstechnik, 58, no. 1, 2010, p. 15–25, ISSN: 0178-2312.

[11] Benner P., Mehrmann V., Sima V., Van Huffel S., Varga A., *SLICOT — A subroutine library in systems and control theory*, Applied and Computational Control, Signals, and Circuits, Datta B. N., Ed., Birkhäuser, Boston, MA, vol. 1, chapter 10, 1999, p. 499–539.

[12] Benner P., Sima V., *Solving algebraic Riccati equations with SLICOT*, Proceedings of The 11th Mediterranean Conference on Control and Automation MED'03, June 18–20 2003, Rhodes, Greece, 2003, 6 pages.

[13] Van Huffel S., Sima V., Varga A., Hammarling S., Delebecque F., *High-performance numerical software for control*, IEEE Control Syst. Mag., **24**, no. 1, 2004, p. 60–76.

[14] Arnold W. F. III, Laub A. J., *Generalized eigenproblem algorithms and software for algebraic Riccati equations*, Proc. IEEE, **72**, no. 12, (1984) 1746–1754.

[15] Laub A. J., *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Contr., AC–24, no. 6, 1979, p. 913–921.

[16] Pappas T., Laub A. J., Sandell N. R., *On the numerical solution of the discrete-time algebraic Riccati equation*, IEEE Trans. Automat. Contr., AC–25, no. 4, 1980, p. 631–641.

[17] Van Dooren P., *A generalized eigenvalue approach for solving Riccati equations*, SIAM J. Sci. Stat. Comput., 2, no. 2, 1981, p. 121–135.

[18] Benner P., Byers R., Losse P., Mehrmann V., Xu H., *Numerical solution of real skew-Hamiltonian/Hamiltonian eigenproblems*, Technische Universität Chemnitz, Chemnitz, Tech. Rep., Nov. 2007.

[19] Benner P., Byers R., Mehrmann V., Xu H., *Numerical computation of deflating subspaces of skew Hamiltonian/Hamiltonian pencils*, SIAM J. Matrix Anal. Appl., **24**, no. 1, 2002, p. 165–190.

[20] Raines A. C. III, Watkins D. S., *A class of Hamiltonian-symplectic methods for solving the algebraic Riccati equation*, Washington State University, Pullman, WA, Tech. Rep., 1992.

[21] Sima V., *Structure-preserving computation of stable deflating subspaces*, Proceedings of the 10th IFAC Workshop Adaptation and Learning in Control and Signal Processing (ALCOSP 2010), Antalya, Turkey, 26–28 August 2010, Copyright 2010 IFAC, 6 pages, IFACPapersOnLine, vol. 10, Part 1, http://www.ifac-papersonline.net/Detailed/46793.html, Identifier 10.3182/20100826-3-TR-4015.00047.

[22] Sima V., *Computational experience with structure-preserving Hamiltonian solvers in optimal control*, Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2011), Noordwijkerhout, The Netherlands, 28–31 July, 2011, Ferrier J.-L., Bernard A., Gusikhin O., Madani K., Eds., SciTePress — Science and Technology Publications, vol. 1 (2011) 91–96, ISBN 978-989-8425-74-4. DOI 10.5220/0003534100910096.

[23] Byers R., *Solving the algebraic Riccati equation with the matrix sign function*, Lin. Alg. Appl., **85**, no. 1, 1987, p. 267–279.

[24] Gardiner J. D., Laub A. J., *A generalization of the matrix sign function solution for algebraic Riccati equations*, Int. J. Control, **44**, 1986, p. 823–832.

[25] Hammarling S. J., *Newton's method for solving the algebraic Riccati equation*, National Physics Laboratory, Teddington, Middlesex TW11 OLW, U.K., NPC Report DIIC 12/82, 1982.

[26] Chu E.-W., Fan H.-Y., Lin W.-W., *A structure-preserving doubling algorithm for continuous-time algebraic Riccati equations*, Lin. Alg. Appl., 386, 2005, p. 55–80.

[27] Guo P.-C., *A modified large-scale structure-preserving doubling algorithm for a large-scale Riccati equation from transport theory*, Numerical Algorithms, 71, no. 3, 2016, p. 541–552, http://dx.doi.org/10.1007/s11075-015-0008-4

[28] Lanzon A., Feng Y., Anderson B. D. O., Rotkowitz M., *Computing the positive stabilizing solution to algebraic Riccati equations with an indefinite quadratic term via a recursive method*, IEEE Trans. Automat. Contr., AC–53, no. 10, 2008, p. 2280–2291.

[29] Benner P., *Contributions to the numerical solution of algebraic Riccati equations and related eigenvalue problems*, Dissertation, Fakultät für Mathematik, Technische Universität Chemnitz–Zwickau, D–09107 Chemnitz, Germany, Feb. 1997.

[30] Benner P., Byers R., *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Trans. Automat. Contr., **43**, no. 1, 1998, p. 101–107.

[31] Kleinman D. L., *On an iterative technique for Riccati equation computations*, IEEE Trans. Automat. Contr., AC–13, 1968, p. 114–115.

[32] Roberts J., *Linear model reduction and solution of the algebraic Riccati equation by the use of the sign function*, Int. J. Control, **32**, 1980, p. 667–687.

[33] Sima V., Benner P., *Experimental evaluation of new SLICOT solvers for linear matrix equations based on the matrix sign function*, Proceedings of 2008 IEEE Multi-conference on Systems and Control; 9th IEEE International Symposium on Computer-Aided Control Systems Design (CACSD), San Antonio, Texas, U.S.A., September 3–5, 2008, Omnipress, 2008, p. 601–606, ISBN: 9 78-1-4244-2221-0.

[34] Penzl T., *LYAPACK Users Guide*, Technische Universität Chemnitz, Sonderforschungsbereich 393, Numerische Simulation auf massiv parallelen Rechnern, Chemnitz, Tech. Rep. SFB393/00–33, Aug. 2000.

[35] Penzl T., *Numerical solution of generalized Lyapunov equations*, Advances in Comp. Math., 8 (1998) 33–48.

[36] Sima V., *Computational experience in solving algebraic Riccati equations*, Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC' 05, 12–15 December 2005, Seville, Spain, Omnipress, 2005, p. 7982–7987.

[37] Sima V., *An efficient Schur method to solve the stabilizing problem*, IEEE Trans. Automat. Contr., AC–26, no. 3, 1981, p. 724–725.

[38] Varga A., *A Schur method for pole assignment*, IEEE Trans. Automat. Contr., AC–26, no. 2, 1981, p. 17–519.

[39] Mehrmann V., Tan E., *Defect correction methods for the solution of algebraic Riccati equations*, IEEE Trans. Automat. Contr., AC–33, no. 7, 1988, p. 695–698.

[40] Ciubotaru B. D., Staroswiecki M., *Comparative study of matrix Riccati equation solvers for parametric faults accommodation*, Proceedings of the 10th European Control Conference, 23-26 August 2009, Budapest, Hungary, 2009, p. 1371–1376.

[41] Leibfritz F., Lipinski W., *Description of the benchmark examples in COMPl$_e$ib*, Department of Mathematics, University of Trier, D–54286 Trier, Germany, Tech. Rep., 2003.

[42] Sima V., Benner P., *A SLICOT implementation of a modified Newton's method for algebraic Riccati equations*, Proceedings of the 14th Mediterranean Conference on Control and Automation MED'06, June 28-30 2006, Ancona, Italy, Omnipress, 6 pages.

[43] Sima V., *Computational experience in solving continuous-time algebraic Riccati equations using standard and modified Newton's method*, Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2013), Reykjavík, Iceland, 29-31 July, 2013, Ferrier J.-L., Gusikhin O., Madani K., Sasiadek J., Eds., vol. 1, SciTePress — Science and Technology Publications, Portugal, 2013, p. 5–16.

[44] Sima V., *Solving SLICOT benchmarks for algebraic Riccati equations by modified Newton's method*, Proceedings of the 17th Joint International Conference on System Theory, Control and Computing (ICSTCC 2013), October 11-13, 2013, Sinaia, Romania, IEEE (2013) 491–496, ISBN: 978-1-4799-2228-4; 978-1-4799-2227-7.

[45] Sima V., Benner P., *Numerical investigation of Newton's method for solving continuous-time algebraic Riccati equations*, Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO-2014), 1-3 September, 2014, Vienna, Austria, Ferrier J.-L., Gusikhin O., Madani K., Sasiadek J., Eds., vol. 1, SciTePress — Science and Technology Publications, Portugal, p. 404–409.

[46] Sima V., *Computational experience with a modified Newton solver for continuous-time algebraic Riccati equations*, Informatics in Control Automation and Robotics, ser. Lecture Notes in Electrical Engineering, Ferrier J.-L., Gusikhin O., Madani K., Sasiadek J., Eds., Springer International Publishing, 325, ch. 3, 2015, p. 55–71, ISBN: 978-3-319-10891-9; 978-3-319-10890-2; ISSN: 1876-1100.

[47] Sima V., *Solving discrete-time algebraic Riccati equations using modified Newton's method*, 6th International Scientific Conference on Physics and Control, San Luis Potosí, Mexico. August 26th-29th, 2013, IPACS Library, 6 pages.

[48] Sima V., Benner P., *Numerical investigation of Newton's method for solving discrete-time algebraic Riccati equations*, Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO-2018), 29-31 July, 2018, Porto, Portugal, Madani K., Gusikhin O., Eds., vol. 1, SciTePress — Science and Technology Publications, Portugal, p. 66–75.

[49] Sima V., *Efficient computations for solving algebraic Riccati equations by Newton's method*, Proceedings of the 2014 18th Joint International Conference on System Theory, Control and Computing (ICSTCC 2014), October 17-19, 2014, Sinaia, Romania, Matcovschi M. H., Ferariu L., Leon F., Eds., IEEE, 2014, p. 609–614, ISSN 978-1-4799-4602-0.

[50] Dongarra J. J., Du Croz J., Duff I. S., Hammarling S., Algorithm 679: *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Softw., **16**, no. 1, 1990, p. 1–17, 18–28.

[51] Van Huffel S., Sima V., *SLICOT and control systems numerical software packages*, Proceedings of the 2002 IEEE International Conference on Control Applications and IEEE International Symposium on Computer Aided Control System Design, CCA/CACSD 2002, September 18–20, 2002, Glasgow, Scotland, U.K., Omnipress, 2002, 39–44, ISBN: 0-7803-7388-X.

[52] Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D., *LAPACK Users' Guide: Third Edition*, ser. Software · Environments · Tools, SIAM, Philadelphia, PA, 1999.

[53] Golub G. H., Van Loan C. F., *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MA, 1996.

[54] Abels J., Benner P., *CAREX—A collection of benchmark examples for continuous-time algebraic Riccati equations (Version 2.0)*, SLICOT Working Note 1999-14, Nov. 1999. Available www.slicot.org

[55] Jónsson G. F., Vavasis S., *Solving polynomials with small leading coefficients*, SIAM J. Matrix Anal. Appl., **26**, no. 2, 2004, p. 400–414.

[56] Armstrong E. S., Rublein G. T., *A stabilization algorithm for linear discrete constant systems*, IEEE Trans. Automat. Contr., AC-21, no. 4, 1976, p. 629–631.