Numerical Investigation of Newton's Method for Solving Discrete-time Algebraic Riccati Equations

Vasile Sima[1] and Peter Benner[2]

[1]*Modelling, Simulation, Optimization Department, National Institute for Research & Development in Informatics, Bd. Mareşal Averescu, Nr. 8–10, Bucharest, Romania*

[2]*Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, Magdeburg, Germany*

Keywords: Algebraic Riccati Equation, Numerical Methods, Optimal Control, Optimal Estimation.

Abstract: A Newton-like algorithm and some line search strategies for solving discrete-time algebraic Riccati equations are discussed. Algorithmic and implementation details incorporated in the developed solver are described. Some numerical results of an extensive performance investigation on a large collection of examples are summarized. These results often show significantly improved accuracy, measured in terms of normalized and relative residuals, in comparison with the state-of-the-art MATLAB function. The new solver is strongly recommended for improving the solutions computed by other solvers.

1 INTRODUCTION

Many procedures for control systems analysis and design require the solution of algebraic Riccati equations (AREs). Such equations appear in various domains and practical applications, including model reduction, optimal filtering, guidance, (robust) control, robotics, etc. Discrete-time AREs (DAREs) are important since many measured systems are modeled by difference equations. Let A, $E \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{n\times m}$, and Q and R be symmetric matrices of suitable dimensions. In a compact notation, the generalized DAREs, with unknown $X = X^T \in \mathbf{R}^{n\times n}$, are defined by

$$\begin{aligned} 0 &= Q + \mathrm{op}(A)^T X\,\mathrm{op}(A) - \mathrm{op}(E)^T X\,\mathrm{op}(E) \\ &\quad -\sigma \mathcal{L}(X)\hat{R}(X)^{-1}\mathcal{L}(X)^T =: \mathcal{R}(X), \end{aligned} \qquad (1)$$

where $\sigma = \pm 1$, E and $\hat{R}(X)$ are nonsingular, and

$$\begin{aligned} \hat{R}(X) &:= R + \sigma B^T X B, \\ \mathcal{L}(X) &:= S + \mathrm{op}(A)^T X B, \end{aligned} \qquad (2)$$

with S of suitable size. The operator $\mathrm{op}(M)$ represents either M or M^T, corresponding to a control or a filtering problem, respectively. A and E are the state and descriptor matrices, respectively, of a linear time-invariant dynamic system, and, in a control setting, B is the input matrix. The use of the $\pm$ sign through σ offers a greater generality. In practice, often Q and S are expressed as $C^T\hat{Q}C$ and $S = C^T\hat{S}$, respectively, where $C \in \mathbf{R}^{p\times n}$ is the output matrix of the system, and C, $\hat{Q}$, and $\hat{S}$ are given.

The solutions of a DARE are the matrices $X = X^T$ for which $\mathcal{R}(X) = 0$. Usually, what is needed is a *stabilizing solution*, X_s, for which the matrix pair $(A - \sigma\,\mathrm{op}(BK(X_s)), E)$ is stable (in a discrete-time sense), where $\mathrm{op}(K(X_s))$ is the gain matrix of the optimal regulator or estimator, given by

$$K(X) := \hat{R}(X)^{-1}\mathcal{L}(X)^T, \qquad (3)$$

with X replaced by X_s. For the dynamic system $Ex_{k+1} = Ax_k + Bu_k$, $k = 0, 1, \ldots$, $x(0) = x_0$, the optimal control trajectory is given by the state feedback law $u_k = -\sigma K(X_s)x_k$. By a proper selection of Q, S, and R, the closed-loop dynamics can be modified to achieve a fast transient response, disturbance rejection, etc. Note that for a filtering problem, B should be replaced by the transpose of C, and the computed $K(X)$ is the transpose of the filter gain matrix. When Y is not a solution of (1), then $\mathcal{R}(Y)$ differs from the zero matrix; $\mathcal{R}(Y)$ is called the *residual* of (1) in Y. The Frobenius norm of $\mathcal{R}(Y)$, $\|\mathcal{R}(Y)\|_F$, is a measure of the error in Y with respect to the solution X.

There is an impressive literature concerning theory and numerical solution of AREs and their practical applications. Several monographs, e.g., (Anderson and Moore, 1971; Bini et al., 2012; Lancaster and Rodman, 1995; Mehrmann, 1991; Sima, 1996) address various theoretical and practical results. Existence and uniqueness results for ARE solutions are dealt with, for instance, in (Lancaster and Rodman, 1980; Lancaster et al., 1986; Lancaster et al., 1987).

Sima, V. and Benner, P.
Numerical Investigation of Newton's Method for Solving Discrete-time Algebraic Riccati Equations.
DOI: 10.5220/0006860600660075
In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2018) - Volume 1*, pages 66-75
ISBN: 978-989-758-321-6

Many "direct" or iterative algorithms have been proposed for solving AREs. The first class includes the (generalized) Schur techniques, e.g., (Arnold and Laub, 1984; Kenney et al., 1989; Laub, 1979; Pappas et al., 1980; Van Dooren, 1981), or structure-exploiting (QR-like) methods, e.g., (Bunse-Gerstner and Mehrmann, 1986; Sima and Benner, 2015; Benner et al., 2016). (These techniques are actually also iterative, but they are applied to a matrix or matrix pencil defined by the given matrices of an ARE.) The second class has several categories, including sign function techniques, e.g., (Balzer, 1980; Byers, 1987; Gardiner and Laub, 1986; Roberts, 1980; Sima and Benner, 2008), Newton techniques, e.g., (Anderson, 1978; Arnold and Laub, 1984; Guo and Laub, 2000; Hammarling, 1982), doubling algorithms, e.g., (Chu et al., 2005; Guo et al., 2006; Guo et al., 2007; Guo, 2016), or recursive algorithms (Lanzon et al., 2008).

Newton's method for solving AREs has been considered by many authors, for instance, (Kleinman, 1968; Hewer, 1971; Arnold and Laub, 1984; Mehrmann, 1991; Lancaster and Rodman, 1995; Sima, 1996; Benner, 1997; Benner, 1998; Benner and Byers, 1998). Moreover, the matrix sign function method for AREs, see (Byers, 1987; Gardiner and Laub, 1986) and the references therein, is actually a specialization of Newton's method for computing the square root of the identity matrix of order $2n$.

Newton's method is best used for iterative improvement of a solution, or as a defect correction method (Mehrmann and Tan, 1988), delivering the maximal possible accuracy when starting from a good approximate solution. Moreover, it may be preferred in implementing certain fault-tolerant or slowly varying systems, which require online controller updating (Ciubotaru and Staroswiecki, 2009); then, the previously computed ARE solution can be used for initialization. Some robotics applications can also benefit from using iterative ARE solvers. For this reason, such algorithms are used in a new open-source C++ library for robotics, optimal and model predictive control (Giftthaler et al., 2018), for solving both continuous-time AREs (CAREs) and DAREs.

This paper summarizes the main theoretical facts about Newton's method for DAREs, as well as implementation issues and numerical results obtained using the newly developed solver. There are several contributions comparing to (Benner, 1998; Benner and Byers, 1998) concerning, e.g., generality and functionality, line search strategies, or stopping criteria. The paper complements our previous studies on the numerical solution of CAREs by Newton's method with line search reported in (Sima and Benner, 2014; Sima, 2015).

2 CONCEPTUAL ALGORITHM

The following **Assumptions** are made:

1. Matrix E is nonsingular.
2. Matrix pair $(\operatorname{op}(E)^{-1}\operatorname{op}(A), \operatorname{op}(E)^{-1}B)$ is stabilizable.
3. Matrix $R = R^T$ is non-negative definite ($R \geq 0$).
4. A stabilizing solution X_s exists and it is unique, and $\hat{R}(X_s)$ is positive definite ($\hat{R}(X_s) > 0$).

Note that Assumption 1 is not actually used by the developed solver, contrary to other solvers (including MATLAB function `dare`).

The algorithmic variants considered in the sequel for DAREs are extensions of Newton's method, which employ a *line search* procedure attempting to reduce the residual along the Newton direction.

The conceptual algorithm can be stated as follows (Benner, 1998):

Algorithm NDARE: Modified Newton method for DARE

Input: The coefficient matrices E, A, B, Q, R, and S, and an initial matrix $X_0 = X_0^T$.
Output: The approximate solution X_k of DARE (1).

```
FOR $k = 0, 1, \ldots, k_{\max}$, DO
  1. Compute $\mathcal{R}(X_k)$.  If convergence or non-convergence is detected, return $X_k$ and/or a warning or error indicator value.
  2. Compute $K_k := K(X_k)$ in (3) and $\operatorname{op}(A_k)$, where $A_k = \operatorname{op}(A) - \sigma B K_k$.
  3. Solve the discrete-time generalized Stein equation
       $\operatorname{op}(A_k)^T N_k \operatorname{op}(A_k) - \operatorname{op}(E)^T N_k \operatorname{op}(E) = -\mathcal{R}(X_k)$   (4)
     for $N_k$.
  4. Find a step size $t_k$ which approximately minimizes $\|\mathcal{R}(X_k + tN_k)\|_F^2$, with respect to $t$.
  5. Update $X_{k+1} = X_k + t_k N_k$.
END
```

Equation (4) is also called discrete-time generalized Lyapunov equation. The usual, standard Lyapunov equation has $E = I_n$.

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, Newton's method with line search often outperforms the standard Newton's method.

If the assumptions above hold and X_0 is stabilizing, then the iterates of the Algorithm NDARE with $\sigma = 1$ and $t_k = 1$ have the following properties (Benner, 1997):

(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) There is a constant $\gamma > 0$ such that

$$\|X_{k+1} - X_s\| \le \gamma \|X_k - X_s\|^2, \quad k \ge 1. \qquad (5)$$

Note that the global quadratic convergence formula (5) does not hold for $k = 0$, involving the iterates X_0 and X_1.

Weaker results are available for the modified Newton algorithm. One such result (Benner, 1997) states that if X_k is stabilizing, then N_k computed by Algorithm NDARE is a descent direction for $\|\mathcal{R}(X_k)\|_F^2$ from X_k, unless $X_k = X_s$.

3 ALGORITHMIC DETAILS

Algorithm NDARE was implemented in a Fortran 77 subroutine, `SG02CD`, following the SLICOT Library (Benner et al., 1999; Benner and Sima, 2003; Benner et al., 2010; Van Huffel et al., 2004) implementation and documentation standards (see http://www.slicot.org). The same routine also solves CAREs. A first version of `SG02CD` and preliminary results on random examples and SLICOT CAREX benchmark collection (Abels and Benner, 1999a) have been presented in (Sima and Benner, 2006). The implemented solver deals with generalized DAREs without inverting the matrix E, which is very important for numerical reasons, since E might be ill-conditioned with respect to inversion. Standard DAREs are solved with the maximal possible efficiency. Moreover, both control and filter DAREs can be solved by the same routine, using an option ("mode") parameter, which specifies the op operator. The matrices A and E are not transposed.

The essential computational procedures involved in Algorithm NDARE are detailed below.

3.1 Removing S Matrix

If R is nonsingular, DAREs can be put in a simpler form, which is more convenient for Newton algorithms. Specifically, setting

$$\tilde{A} = A - \sigma\, \mathrm{op}(BR^{-1}S^T), \quad \tilde{Q} = Q - \sigma S R^{-1} S^T, \quad (6)$$

after redefining A and Q as $\tilde{A}$ and $\tilde{Q}$, respectively, equation (1) reduces to

$$\begin{aligned} 0 = {} & \mathrm{op}(A)^T X\, \mathrm{op}(A) - \mathrm{op}(E)^T X\, \mathrm{op}(E) \\ & - \sigma\, \mathrm{op}(A)^T X \hat{G}(X) X\, \mathrm{op}(A) + Q =: \mathcal{R}(X), \quad (7) \end{aligned}$$

where $\hat{G}(X) := B\hat{R}(X)^{-1}B^T$, and the second term reduces to X in the standard case ($E = I_n$). The transformations in (6) eliminate the matrix S from the formulas to be used. In this case, the matrix K_k may sometimes no longer be computed in Step 2, and $A_k = \mathrm{op}(A) - \sigma \hat{G}_k X_k\, \mathrm{op}(A)$, with $\hat{G}_k := \hat{G}(X_k)$.

To obtain $\tilde{A}$ and $\tilde{Q}$ in (6), the Cholesky factor of R, R_c, can be used if $R > 0$, where $R =: R_c^T R_c$, with R_c upper triangular. Defining $\tilde{B} = BR_c^{-1}$ and $\tilde{S} = SR_c^{-1}$, the relations (6) are equivalent to

$$\tilde{A} = A - \sigma\, \mathrm{op}(\tilde{B}\tilde{S}^T), \quad \tilde{Q} = Q - \sigma \tilde{S}\tilde{S}^T, \qquad (8)$$

so just two triangular systems need to be solved, and two matrix products are computed for obtaining $\tilde{A}$ and $\tilde{Q}$, after factoring R. Symmetry is exploited for getting $\tilde{Q}$ via a BLAS (Dongarra et al., 1990) symm operation. Similarly, if $\hat{R}(X_k) > 0$, then the Cholesky factor of $\hat{R}(X_k)$, $\hat{R}_c(X_k)$, can be used. Defining $D_k := D(X_k) := B\hat{R}_c(X_k)^{-1}$, then $\hat{G}_k = D_k D_k^T$, and $A_k = \mathrm{op}(A) - \sigma D_k D_k^T X_k\, \mathrm{op}(A)$. The use of D_k instead of $\hat{G}_k$ is convenient when m is sufficiently smaller than n ($m \le n/4$). If $\hat{G}_k$ is to be preferred (since $m > n/4$), but the norm of $\hat{G}_0$ is too large, then, if possible, the factor D_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.

When R is not positive definite, then either UDU^T or LDL^T factorization (Golub and Van Loan, 1996) 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.

3.2 Using S Matrix

When $S \ne 0$, but R is ill-conditioned with respect to inversion, the use of formulas (6) will potentially introduce large errors from the beginning of the iterative process, which will be propagated during the entire process. This might involve a degradation of its behavior, resulting in slower convergence, and/or an inaccurate computed solution. Using S during the iterative process could avoid such degradation. Therefore, an option of the solver allows to avoid the transformations (6), and involve S in all subsequent calculations. In this case, other formulas are needed, since $\hat{G}_k$ cannot be used. Specifically, define

$$H_k := \mathrm{op}(A_k)^T X_k B + S, \quad F_k = H_k \hat{R}_c(X_k)^{-1}, \quad (9)$$

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

$$\begin{aligned} \mathcal{R}(X_k) &= Q + \mathrm{op}(A)^T X_k\, \mathrm{op}(A) \\ &\quad - \mathrm{op}(E)^T X_k\, \mathrm{op}(E) - \sigma F_k F_k^T, \quad (10) \\ A_k &= \mathrm{op}(A) - \sigma D_k F_k^T, \quad (11) \end{aligned}$$

where D_k has been defined above.

If, however, $\hat{R}(X_k)$ is indefinite, then the needed formulas follow directly from (1)–(3), namely,

$$\begin{aligned}\mathcal{R}(X_k) &= Q + \mathrm{op}(A)^T X_k \,\mathrm{op}(A) \\ &\quad - \mathrm{op}(E)^T X_k \,\mathrm{op}(E) - \sigma H_k K_k, \quad (12) \\ A_k &= \mathrm{op}(A) - \sigma B K_k, \quad (13)\end{aligned}$$

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

The implementation is optimized by using common subexpressions when computing $\mathcal{R}(X_k)$ and $\mathrm{op}(A_k)$, taking also into account the ratio between n and m. Various formulas for efficient implementation of Newton's method for AREs are proven in (Sima, 2014).

3.3 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 NDARE could converge to another DARE 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.

Another 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 strategy.

Optionally, the matrices A_k and E (if E is general) are scaled for solving the Stein equations, and their solutions are suitably updated. Note that the LAPACK subroutines DGEES and DGGES (Anderson et al., 1999), which are called by the standard and generalized Stein 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, and no scaling, the convergence was not achieved in a reasonable number of iterations. Moreover, sometimes scaling allows to compute more accurate solutions and/or use less iterations, comparing to the case with no scaling.

A maximum allowed number of iteration steps, $k_{\max}$, is specified on input, and the number of iteration steps performed, k_s, is returned on exit.

3.4 Finding the Step Size

The optimal step size t_k is given by

$$t_k = \arg\min_t \|\mathcal{R}(X_k + tN_k)\|_F^2. \quad (14)$$

Since solving (14) for a DARE is expensive, it was suggested in (Benner, 1997; Benner, 1998) to use an approximate value t_k to be found numerically as the argument of the minimal value in [0,2] of a polynomial of order 4. Indeed,

$$\mathcal{R}(X_k + tN_k) = (1-t)\mathcal{R}(X_k) - t^2 V_k, \quad (15)$$

where $V_k = \mathrm{op}(A_k)^T N_k \hat{G}_k N_k \,\mathrm{op}(A_k)$. The problem (14) is replaced by the minimization of the approximate quartic polynomial (Benner, 1997)

$$\begin{aligned} f_k(t) &:= \mathrm{trace}(\mathcal{R}(X_k + tN_k)^2) \\ &\approx \alpha_k(1-t)^2 - 2\beta_k(1-t)t^2 + \gamma_k t^4, \quad (16)\end{aligned}$$

where $\alpha_k = \mathrm{trace}(\mathcal{R}(X_k)^2)$, $\beta_k = \mathrm{trace}(\mathcal{R}(X_k)V_k)$, $\gamma_k = \mathrm{trace}(V_k^2)$.

To solve this problem, a cubic polynomial (the derivative of $f_k(t)$) is set up, whose 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 (Jónsson and Vavasis, 2004).

Actually, the true $f_k(t)$ for DAREs is a rational function, and the above formulas are obtained by replacing its denominator by the second order Taylor series approximant at $t = 0$. The approximation is useful when t is small enough. For instance, if $t < 1/\|\hat{G}_k N_k\|$, 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 $\|\hat{G}_k N_k\| < 1/2$. It can be shown (Benner, 1997) that if X_k is stabilizing, then either N_k is a descent direction for $\|\mathcal{R}(X_k)\|_F^2$, 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 step sizes t_k could be too small, and the progress of the iteration could be too slow.

3.5 Iterative Process

The algorithm computes the initial residual matrix $\mathcal{R}(X_0)$ and the matrix $\mathrm{op}(A_0)$, where $A_0 := \mathrm{op}(A) - \sigma\hat{G}_0 X_0 \,\mathrm{op}(A)$. If no initial matrix X_0 is given, then $X_0 = 0$, $\mathcal{R}(X_0) = \tilde{Q}$ and $\mathrm{op}(A_0) = \tilde{A}$ in (6).

At the beginning of the iteration k, $0 \le k \le k_{\max}$, 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. If $\min(r_k, r_r(X_k)) > \tau$, a standard (if $E = I_n$) or generalized Stein equation (4) is solved for N_k (the Newton direction).

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

$$r_k := \|\mathcal{R}(X_k)\|_F / \max(1, \|X_k\|_F) \leq \tau, \tag{17}$$

the iterative process is successfully terminated. If $\tau \leq 0$, a default tolerance is used, defined in terms of the Frobenius norms of the given matrices, and relative machine precision, ε_M. Specifically, τ is computed by the formula

$$\begin{aligned}\tau \;=\; \min\{\,& \varepsilon_M\sqrt{n}\big(\|A\|_F\,(\|A\|_F + \|D_0\|_F^2\,\|A\|_F) \\ & + \|E\|_F^2 + \|Q\|_F\big), \sqrt{\varepsilon_M}/10^3\,\}.\end{aligned} \tag{18}$$

(The factor $\|D_0\|_F^2$ is replaced by $\hat{G}_0$ if $\hat{R}(X_0)$ is indefinite.) The second operand of min in (18) was introduced to prevent deciding convergence too early for systems with very large norms for A, E, D_0 (or $\hat{G}_0$), and/or Q.

For systems with very large norms of the matrices A, E, D_0 (or $\hat{G}_0$), and/or Q, and small norm of the solution X, the termination criterion involving (18) might not be satisfied in a reasonable number of iterations (or never, due to accumulated rounding errors), while 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 four matrix terms of (1) in the denominator of its formula, is also tested at iterations $10 + 5q$, $q = 0, 1, \ldots$, and it might produce the termination of the iterative process, instead of the criterion based on the normalized residual. The relative residual is not tested at each iteration in order to reduce the computation costs, and to increase the chances of termination via the normalized residual test.

Another test is to check 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, set $X_{k+1} = X_k + t_k N_k$, and compute the updated matrices $\mathrm{op}(A_{k+1})$ and $\mathcal{R}(X_{k+1})$. 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 $\mathcal{R}(X_k + t_k N_k)$ can be efficiently performed by updating the residual $\mathcal{R}(X_k)$, the original data is used, since the updating formula (15) could suffer from severe numerical cancellation, and hence it could compromise the accuracy of the intermediate results.

Often, but 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 $t_k < 0.5$, $\varepsilon_M^{1/4} < r_k < 1$, and $\|\mathcal{R}(X_k + t_k N_k)\|_F \leq 10$, if this happens during the first 10 iterations. The rationale of this strategy is that if the residual is small enough after the first few iterations, the use of a standard Newton step could reduce the residual faster than a Newton algorithm with small step sizes.

In order to detect stagnation, the last computed k_B residuals are stored in an array RES. If $\|\mathcal{R}(X_k + t_k N_k)\|_F > \tau_s\|\mathcal{R}(X_{k-k_B})\|_F > 0$, 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.

3.6 Line Search Strategies

Other three line search stategies may be chosen besides the *pure line search strategy*, which uses a solution t_k of the approximate quartic polynomial (16) 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. The rationale for this strategy is that when the normalized residual is small enough, line search cannot offer sensible improvements, and the standard algorithm converges with a fast rate, usually quadratrically as to be expected from the local convergence theory of Newton's method. In addition, in such an instance, t_k will be close to 1, and typically there will be no difference between the values of $\|\mathcal{R}(X_k)\|_F$ computed for t_k and for 1. Therefore, the calculations for finding t_k are avoided.

In the *hybrid strategy*, both standard Newton step and the step corresponding to the approximate line search procedure are computed, and that step which gives the smallest residual is selected at each iteration. Finally, the *backtracking strategy*, proposed in (Benner, 1997), is a special hybrid strategy in which the selected step is only taken provided there is a sufficient residual 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.

3.7 Memory Storage Issues

The arrays holding the data matrices A and E are unchanged on exit, except when $S \neq 0$, but it should and could be removed from DARE using (6). In this special case, $\tilde{A}$ is returned. Array `Q` stores matrix Q on entry and the computed solution X_s on exit. If $m \leq n/4$ and the Cholesky factor $\hat{R}_c(X_s)$ can be computed, then the array `B`, storing B on input, returns the final matrix $D(X_s)$. Otherwise, array `B` is unchanged on exit. Similarly, the array `R`, storing R on input, may return either the Cholesky factor, if it can be computed, or the factors of the UDU^T or LDL^T factorization of $\hat{R}(X_s)$, if $\hat{R}(X_s)$ 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(k_s, 50)$+1$ values of the residuals, 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, if used, $\hat{G}_k$ 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 $\mathcal{R}(X_k)$ are concatenated along the main diagonals in a two-dimensional $n(n+1)$ array, and similarly for $\hat{G}_k$ and a copy of the matrix Q, if $\hat{G}_k$ is used. Array `Q` itself is also used for temporarily storing the residual matrix $\mathcal{R}(X_k)$, 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 new solver 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). A MATLAB executable MEX-function has been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

Besides tests with randomly generated matrices, the results for which are not reported here, other tests have been conducted for linear systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2004). Preliminary results have been presented in (Sima, 2013a; Sima, 2013b). (The second reference summarizes the results obtained using Newton’s method for solving AREs for examples from the SLICOT benchmark collections for CAREs (Abels and Benner, 1999a) and DAREs (Abels and Benner, 1999b).)

The COMPl$_e$ib collection contains 124 standard continuous-time examples (with $E = I_n$), with several variations, giving a total of 168 problems. For testing purposes, these examples have been considered in this paper as being of discrete-time type. The performance index matrices Q and R have been chosen as identity matrices of suitable sizes. The matrix S was always zero. All but 16 problems (for systems of order larger than 2000, with matrices in sparse format) have been tried. However, 63 problems did not satisfy the needed conditions for the existence of a stabilizing solution, and could not be solved by the MATLAB function `dare`, which gave the error message “There is no finite stabilizing solution”. These examples have been omitted. In addition, other five 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 HF2D_CD4 and HF2D_CD6, respectively. Such matrices proved to offer a poor initialization for Newton’s method.

In a series of tests, X_0 was set to a zero matrix, if A was found to be stable; otherwise, an initialization of the Newton solver with a matrix computed using the stabilization algorithm in (Armstrong and Rublein, 1976) was tried, and when this algorithm failed to deliver a stabilizing X_0 matrix, the solution provided by `dare` was used. A zero initialization could be tried for 7 stable examples, namely AC5, REA4, BDT1, CSE1, TMD, FS, and ROC5, but the Newton solver failed for CSE1 with $X_0 = 0$, since a singular Stein equation was found. The stabilization algorithm was tried on 82 unstable systems, and succeeded for 55 examples, hence it failed for 27 examples. Both standard and modified Newton’s method, with or without balancing the coefficient matrices of the Stein equations, were tried.

Tests with X_0 computed by the stabilization algorithm also for stable systems, or with X_0 returned by MATLAB `dare` for all examples, have also been successfully performed. The last set of tests shows the performance of the Newton solver in refining a solution computed by another solver.

A brief selection of results is presented below. For standard Newton's method with `dare` initialization, nonzero differences in the normalized residuals for default and ε_M tolerance values were encountered for 16 COMPl$_e$ib examples, and they were of the same order as, or lower order of magnitude than the residuals themselves. The number of iterations for the tolerance ε_M increased by 1 (for six examples), 2 (for two examples), 3 (for three examples), but also by 10 (for DLR1), 24 (for HE6 and HE7), and by 39 and 48 (for NN11, and AGS, respectively). This shows that with `dare` initialization, it is preferable to use the default tolerance, since a too small value, such as ε_M, will eventually reduce the residuals only marginally, but possibly after many more iterations. Actually, for HE6, HE7, NN11, and AGS, the normalized residuals slightly increased for a tolerance set to ε_M. The solution computed by `dare` had a very large Frobenius norm, of order 10^{10} or larger, for HE6, HE7, AGS, NN11, and DLR1, but also for PAS, and of order 10^8 and 10^7 for HF2D_IS7 and HF2D_CD5, respectively.

Figure 1 displays the normalized residuals for examples from the COMPl$_e$ib collection, using MATLAB function `dare` and the standard Newton solver, with default tolerance and `dare` initialization. With few exceptions, the Newton solver is either comparable with `dare` or it improved the normalized residuals, sometimes by several orders of magnitude. However, for four examples (HF2D_IS7, HF2D_CD5, HF2D17, and HF2D18, numbered as 59, 61, 69, and 70, respectively, in Fig. 1), clearly worse results have been obtained. Line search succeeded to get smaller normalized residuals for these examples, as can be seen in Fig. 2.

Figure 3 plots the MATLAB-style relative residuals for examples from the COMPl$_e$ib collection, using MATLAB function `dare` and Newton solver with line search, with default tolerance and `dare` initialization. The Newton solver returned comparable or (much) smaller residuals except for three examples, namely, HF2D_IS7, HF2D17, and HF2D18 (numbered as 59, 69, and 70, respectively). For the last two examples, the standard method gave smaller residuals than the line search method.

Similarly, Fig. 4 shows the corresponding elapsed CPU times for the two solvers. For 18 examples, the

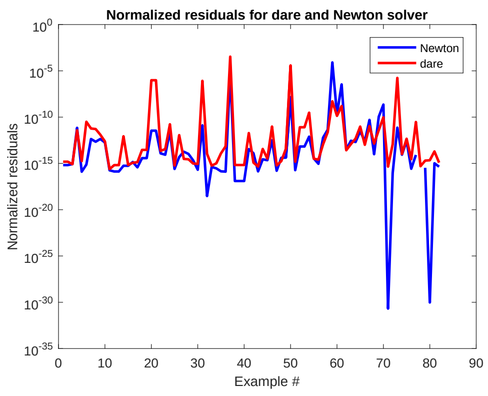

Figure 1: Normalized residuals for examples from the COMPl$_e$ib collection (taken as discrete-time systems), using MATLAB function `dare` and standard Newton solver, with default tolerance and `dare` initialization.

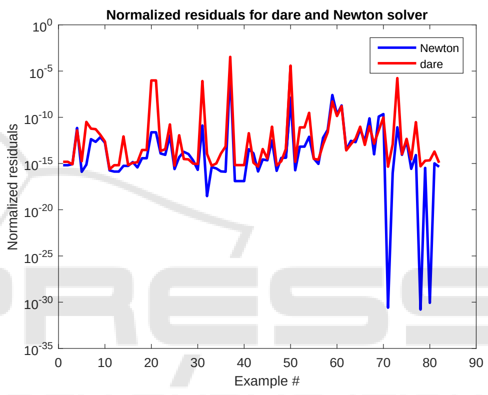

Figure 2: Normalized residuals for examples from the COMPl$_e$ib collection, using MATLAB function `dare` and Newton solver with line search, default tolerance and `dare` initialization.

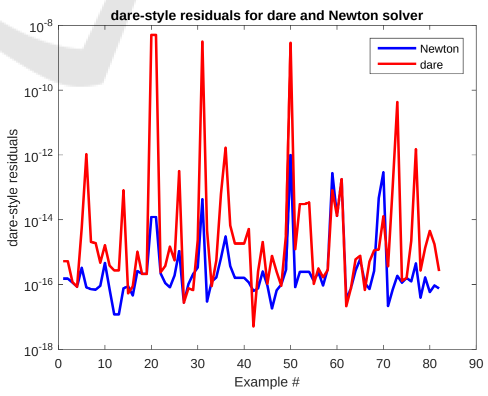

Figure 3: MATLAB-style residuals for examples from the COMPl$_e$ib collection, using MATLAB function `dare` and Newton solver with line search, default tolerance and `dare` initialization.

computations with standard Newton method ended before finishing the first iteration, and just six examples (AGS, PAS, NN11, HF2D_IS7, HF2D_CD5, and

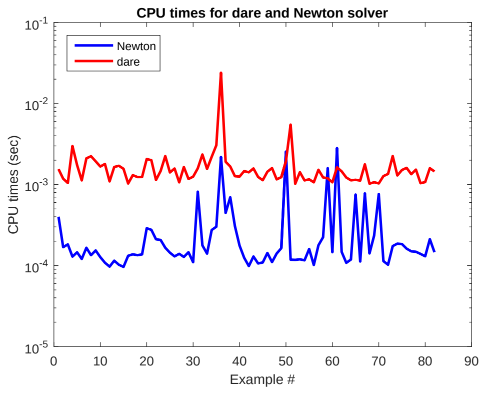

Figure 4: Elapsed CPU time for examples from the COMPl$_e$ib collection, using MATLAB function `dare` and standard Newton solver, with default tolerance and `dare` initialization.

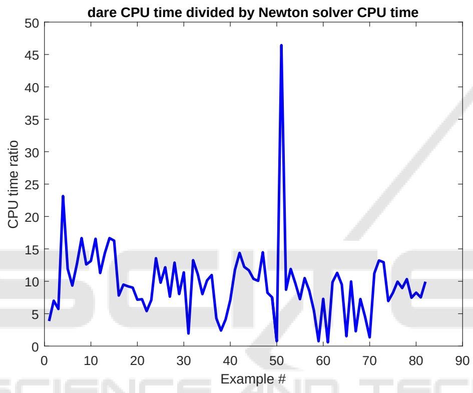

Figure 5: Ratios of the elapsed CPU time needed by MATLAB function `dare` and standard Newton solver, with default tolerance and `dare` initialization, for examples from the COMPl$_e$ib collection.

HF2D17) needed more than one iteration, namely, 8, 11, 11, 50, 50, and 2 iterations, respectively. For the same examples, the modified Newton method needed 2, 11, 11, 11, 0, and 1 iterations, and it was by three and two orders of magnitude more accurate for HF2D_IS7 and HF2D_CD5, respectively, and comparable for all other examples. Since very few iterations are most often needed, the CPU time for the Newton solver is a small fraction of that for the MATLAB solver `dare`. Figure 5 plots the ratios of the elapsed CPU time needed by MATLAB function `dare` and the standard Newton solver.

The bar graph from Fig. 6 shows the improvement obtained using the Newton solver with line search, default tolerance and `dare` initialization. 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, in comparison to `dare`. The number of examples in the six bins are 48, 19, 7, 2, 5,

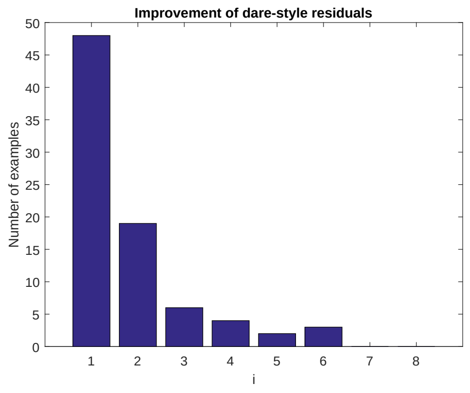

Figure 6: Bar graph showing the improvement of the MATLAB-style residuals for examples from the COMPl$_e$ib collection, using Newton solver with line search, default tolerance and `dare` initialization. The height of the i-th vertical bar indicates the number of examples for which the improvement was between i-1 and i orders of magnitude.

and 1, corresponding to improvements till one order of magnitude, between one and two orders of magnitude, and so on.

5 CONCLUSIONS

Basic facts and improved procedures and algorithms for solving discrete-time algebraic Riccati equations using standard or modified Newton's method, with several line search strategies, have been presented. Numerical results obtained on a comprehensive set of examples from the COMPl$_e$ib collection, taken as discrete-time systems, have been summarized and they illustrate the performance and capabilities of this solver. The possibility to offer, in few iterations, a reduction by one or more orders of magnitude of the normalized and MATLAB-style residuals of the solutions computed by MATLAB function `dare`, makes Newton solver an attractive support tool for solving DAREs.

ACKNOWLEDGEMENTS

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, "New research in complex systems modelling and optimization with applications in industry, business and cloud computing", funded by the Ministry of Research and Innovation, Romania.

REFERENCES

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

Abels, J. and Benner, P. (1999b). DAREX—A collection of benchmark examples for discrete-time algebraic Riccati equations (Version 2.0). SLICOT Working Note 1999-16. Available: www.slicot.org.

Anderson, B. D. O. (1978). Second-order convergent algorithms for the steady-state Riccati equation. *Int. J. Control*, 28(2):295–306.

Anderson, B. D. O. and Moore, J. B. (1971). *Linear Optimal Control*. Prentice-Hall, Englewood Cliffs, New Jersey.

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

Armstrong, E. S. and Rublein, G. T. (1976). A stabilization algorithm for linear discrete constant systems. *IEEE Trans. Automat. Contr.*, AC-21(4):629–631.

Arnold, III, W. F. and Laub, A. J. (1984). Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proc. IEEE*, 72(12):1746–1754.

Balzer, L. A. (1980). Accelerated convergence of the matrix sign function method of solving Lyapunov, Riccati and other matrix equations. *Int. J. Control*, 32(6):1057–1078.

Benner, P. (1997). *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.

Benner, P. (1998). Accelerating Newton's method for discrete-time algebraic Riccati equations. In Beghi, A., Finesso, L., and Picci, G., editors, *Mathematical Theory of Networks and Systems, Proceedings of the MTNS-98 Symposium held in Padova, Italy, July, 1998*, 569–572. Il Poligrafo, Padova, Italy.

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

Benner, P., Kressner, D., Sima, V., and Varga, A. (2010). Die SLICOT-Toolboxen für Matlab. *at—Automatisierungstechnik*, 58(1):15–25.

Benner, P., Mehrmann, V., Sima, V., Van Huffel, S., and Varga, A. (1999). SLICOT — A subroutine library in systems and control theory. In Datta, B. N., editor, *Applied and Computational Control, Signals, and Circuits*, vol. 1, ch. 10, 499–539. Birkhäuser, Boston.

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

Benner, P., Sima, V., and Voigt, M. (2016). Algorithm 961: Fortran 77 subroutines for the solution of skew-Hamiltonian/Hamiltonian eigenproblems. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):1–26.

Bini, D. A., Iannazzo, B., and Meini, B. (2012). *Numerical Solution of Algebraic Riccati Equations*. SIAM, Philadelphia.

Bunse-Gerstner, A. and Mehrmann, V. (1986). A symplectic QR like algorithm for the solution of the real algebraic Riccati equation. *IEEE Trans. Automat. Contr.*, AC–31(12):1104–1113.

Byers, R. (1987). Solving the algebraic Riccati equation with the matrix sign function. *Lin. Alg. Appl.*, 85(1):267–279.

Chu, E.-W., Fan, H.-Y., and Lin, W.-W. (2005). A structure-preserving doubling algorithm for continuous-time algebraic Riccati equations. *Lin. Alg. Appl.*, 386:55–80.

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

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 18–28.

Gardiner, J. D. and Laub, A. J. (1986). A generalization of the matrix sign function solution for algebraic Riccati equations. *Int. J. Control*, 44:823–832.

Giftthaler, M., Neunert, M., Stäuble, M., and Buchli, J. (2018). The control toolbox — An open-source C++ library for robotics, optimal and model predictive control. [Online]. Available: https://arxiv.org/abs/1801.04290.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MA, 3rd edition.

Guo, C. and Laub, A. J. (2000). On a Newton-like method for solving algebraic Riccati equations. *SIAM J. Matrix Anal. Appl.*, 21(2):694–698.

Guo, C.-H., Iannazzo, B., and Meini, B. (2007). On the doubling algorithm for a (shifted) nonsymmetric algebraic Riccati equation. *SIAM J. Matrix Anal. Appl.*, 29(4):1083–1100.

Guo, P.-C. (2016). A modified large-scale structure-preserving doubling algorithm for a large-scale Riccati equation from transport theory. *Numerical Algorithms*, 71(3):541–552.

Guo, X.-X., Lin, W.-W., and Xu, S.-F. (2006). A structure-preserving doubling algorithm for nonsymmetric algebraic Riccati equation. *Numer. Math.*, 103(3):393–412.

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

Hewer, G. A. (1971). An iterative technique for the computation of the steady state gains for the discrete optimal regulator. *IEEE Trans. Automat. Contr.*, AC–16(4):382–384.

Jónsson, G. F. and Vavasis, S. (2004). Solving polynomials with small leading coefficients. *SIAM J. Matrix Anal. Appl.*, 26(2):400–414.

Kenney, C., Laub, A. J., and Wette, M. (1989). A stability-enhancing scaling procedure for Schur-Riccati solvers. *Systems Control Lett.*, 12:241–250.

Kleinman, D. L. (1968). On an iterative technique for Riccati equation computations. *IEEE Trans. Automat. Contr.*, AC–13:114–115.

Lancaster, P., Ran, A. C. M., and Rodman, L. (1986). Hermitian solutions of the discrete algebraic Riccati equation. *Int. J. Control*, 44:777–802.

Lancaster, P., Ran, A. C. M., and Rodman, L. (1987). An existence and monotonicity theorem for the discrete algebraic matrix Riccati equation. *Lin. and Multil. Alg.*, 20:353–361.

Lancaster, P. and Rodman, L. (1980). Existence and uniqueness theorems for the algebraic Riccati equation. *Int. J. Control*, 32:285–309.

Lancaster, P. and Rodman, L. (1995). *The Algebraic Riccati Equation*. Oxford University Press, Oxford.

Lanzon, A., Feng, Y., Anderson, B. D. O., and Rotkowitz, M. (2008). Computing the positive stabilizing solution to algebraic Riccati equations with an indefinite quadratic term via a recursive method. *IEEE Trans. Automat. Contr.*, AC–53(10):2280–2291.

Laub, A. J. (1979). A Schur method for solving algebraic Riccati equations. *IEEE Trans. Automat. Contr.*, AC–24(6):913–921.

Leibfritz, F. and Lipinski, W. (2004). *COMPl$_e$ib* 1.0 – User manual and quick reference. Technical report, Department of Mathematics, University of Trier, Trier, Germany.

Mehrmann, V. (1991). *The Autonomous Linear Quadratic Control Problem. Theory and Numerical Solution*, volume 163 of *Lect. Notes in Control and Information Sciences* (M. Thoma and A. Wyner, eds.). Springer-Verlag, Berlin.

Mehrmann, V. and Tan, E. (1988). Defect correction methods for the solution of algebraic Riccati equations. *IEEE Trans. Automat. Contr.*, AC–33(7):695–698.

Pappas, T., Laub, A. J., and Sandell, N. R. (1980). On the numerical solution of the discrete-time algebraic Riccati equation. *IEEE Trans. Automat. Contr.*, AC–25(4):631–641.

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

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

Sima, V. (2013a). Solving discrete-time algebraic Riccati equations using modified Newton's method. In *6th International Scientific Conference on Physics and Control*, San Luis Potosí, Mexico. August 26th-29th, 2013.

Sima, V. (2013b). Solving SLICOT benchmarks for algebraic Riccati equations by modified Newton's method. In *Proceedings of the 17th Joint International Conference on System Theory, Control and Computing (ICSTCC 2013)*, October 11-13, 2013, Sinaia, Romania, 491–496. IEEE.

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

Sima, V. (2015). Computational experience with a modified Newton solver for continuous-time algebraic Riccati equations. In Ferrier, J.-L., Gusikhin, O., Madani, K., and Sasiadek, J., editors, *Informatics in Control Automation and Robotics*, volume 325 of *Lecture Notes in Electrical Engineering*, ch. 3, 55–71. Springer International Publishing.

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

Sima, V. and Benner, P. (2008). Experimental evaluation of new SLICOT solvers for linear matrix equations based on the matrix sign function. In *Proceedings of 2008 IEEE Multi-conference on Systems and Control. 9th IEEE International Symposium on Computer-Aided Control Systems Design (CACSD)*, San Antonio, TX, U.S.A., September 3–5, 2008, 601–606. Omnipress.

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

Sima, V. and Benner, P. (2015). Solving SLICOT benchmarks for continuous-time algebraic Riccati equations by Hamiltonian solvers. In *Proceedings of the 2015 19th International Conference on System Theory, Control and Computing (ICSTCC 2015)*, October 14-16, 2015, Cheile Gradistei - Fundata Resort, Romania, 1–6. IEEE.

Van Dooren, P. (1981). A generalized eigenvalue approach for solving Riccati equations. *SIAM J. Sci. Stat. Comput.*, 2(2):121–135.

Van Huffel, S., Sima, V., Varga, A., Hammarling, S., and Delebecque, F. (2004). High-performance numerical software for control. *IEEE Control Syst. Mag.*, 24(1):60–76.