Computational Experience in Solving Continuous-time Algebraic Riccati Equations using Standard and Modified Newton's Method

Vasile Sima
Advanced Research, National Institute for Research & Development in Informatics,
Bd. Mareşal Averescu, Nr. 8–10, Bucharest, Romania

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

Abstract: Improved algorithms for solving continuous-time algebraic Riccati equations using Newton's method with or without line search are discussed. The basic theory and Newton's algorithms are briefly presented. Algorithmic details the developed solvers are based on, the main computational steps (finding the Newton direction, finding the Newton step size), and convergence tests are described. The main results of an extensive performance investigation of the solvers based on Newton's method are compared with those obtained using the widely-used MATLAB solver. Randomly generated systems with orders till 2000, as well as the systems from a large collection of examples, are considered. The numerical results often show significantly improved accuracy, measured in terms of normalized and relative residuals, and greater efficiency than the MATLAB solver. The results strongly recommend the use of such algorithms, especially for improving the solutions computed by other solvers.

1 INTRODUCTION

The numerical solution of algebraic Riccati equations (AREs) is an essential step in many computational methods for model reduction, filtering, and controller design for linear control systems. 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 continuous-time AREs (CAREs), with unknown $X = X^T \in \mathbf{R}^{n\times n}$, are defined by

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

where E and R are assumed to be nonsingular, and

$$\mathcal{L}(X) := L + \mathrm{op}(E)^T X B\,,$$

with L of suitable size. The operator $\mathrm{op}(M)$ represents either M or M^T. Define also $G := BR^{-1}B^T$. An optimal regulator problem involves the solution of an ARE with $\mathrm{op}(M) = M$; an optimal estimator problem involves the solution of an ARE with $\mathrm{op}(M) = M^T$, *input matrix* B replaced (by duality) by the transpose of the *output matrix* $C \in \mathbf{R}^{p\times n}$, and m replaced by p. (This means that L should be $n \times p$ in this case.) In practice, often Q and L are given as $C^T\bar{Q}C$ and $L = C^T\bar{L}$, respectively. The solutions of an ARE 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 $(\mathrm{op}(A - BK(X_s)), \mathrm{op}(E))$ is stable (in a continuous-time sense), where $\mathrm{op}(K(X_s))$ is the gain matrix of the optimal regulator or estimator, and

$$K(X) := R^{-1}\mathcal{L}(X)^T \tag{2}$$

(with X replaced by X_s).

There is a vast literature concerning AREs and their use for solving optimal control and estimation problems; see, e.g., the monographs (Anderson and Moore, 1971; Mehrmann, 1991; Lancaster and Rodman, 1995) for many theoretical results. The optimization criterion for linear control systems is a quadratic performance index in terms of the system state and control input. By minimizing this criterion, a solution to the optimal systems stabilization and control is obtained, expressed as a state-feedback control law. 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. It is worth to say that the results of an optimal design are often better suited in practice than those found by other approaches. For instance, pole assignment may deliver too large gain

Sima V..
Computational Experience in Solving Continuous-time Algebraic Riccati Equations using Standard and Modified Newton's Method.
DOI: 10.5220/0004482500050016
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 5-16
ISBN: 978-989-8565-70-9

matrices, producing high-magnitude inputs, which might not be acceptable. In both control and estimation problems, including those stated in the H_∞ theory (e.g., (Francis, 1987)), a major computational step is the solution of an ARE. Due to their importance, numerous numerical methods have been proposed for solving AREs; see, for instance, (Mehrmann, 1991; Sima, 1996). There are also several highly-used software implementation, e.g., in MATLAB (MATLAB, 2011), or in the SLICOT Library (Benner et al., 1999; Benner and Sima, 2003; Van Huffel et al., 2004; Benner et al., 2010).

Newton's method for solving AREs has been considered by many authors, for instance, (Kleinman, 1968; Mehrmann, 1991; Lancaster and Rodman, 1995; Sima, 1996; Benner, 1997; Benner, 1998; Benner and Byers, 1998). Actually, the matrix sign function method for solving AREs, e.g., (Roberts, 1980; Gardiner and Laub, 1986; Byers, 1987; Sima and Benner, 2008), uses a specialized Newton's method to compute the square root of the identity matrix of order $2n$. This paper merely reports on implementation details and numerical results. In addition, there are contributions compared to (Benner, 1998; Benner and Byers, 1998): improved stopping criteria, improved functionality (regarding generality in the coefficient matrices and options), a better routine for computing the roots of a third order polynomial, etc. The paper extends the results of (Sima and Benner, 2006) in some details, and by investigating the numerical behavior of the current Newton-based ARE solvers for high-order random systems, and for systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003; Leibfritz and Lipinski, 2004). (The previous paper (Sima and Benner, 2006) used randomly generated systems with $n \leq 40$, and systems from the CAREX benchmark collection (Abels and Benner, 1999), where most problems have small size, but may be very ill-conditioned.) It is worth mentioning that Newton's method has been applied in (Penzl, 2000) for solving special classes of large-order AREs, using low rank Cholesky factors of the solutions of the Lyapunov equations built during the iterative process (Penzl, 1998). Additional numerical results, for randomly generated systems with $n \leq 600$, and comparison with MATLAB and SLICOT solvers are presented in (Sima, 2005). However, contrary to standard solvers, the specialized solvers used (`lp_lrnm` and `lp_lrnm_i`) are not general solvers. In order to use them advantageously, the following main assumptions must be fulfilled: 1) the matrix A is structured or sparse; 2) the solution X has a small rank in comparison with n. (These solvers use the possibly sparse structure of the matrix A and operations of the form Ab or $A^{-1}b$, where b is a vector.) The solvers discusssed in this paper are general, and can be used to solve large dense problems.

The paper compares the performance of the Newton solver with or without line search with the performance of the state-of-the-art commercial solver `care` from MATLAB Control System Toolbox. The MATLAB solver uses a different, eigenvalue approach, based on the results in, e.g., (Laub, 1979; Van Dooren, 1981; Arnold and Laub, 1984). Relatively recent research, including both theoretical and numerical investigation, has been directed to exploit the Hamiltonian-symplectic structure of the eigenproblem associated to the ARE (Raines and Watkins, 1992; Benner et al., 2002; Benner et al., 2007; Sima, 2010; Sima, 2011).

A recursive method for computing the positive definite stabilizing solution of an ARE with an indefinite quadratic term has been recently proposed in (Lanzon et al., 2008).

One drawback of the Newton's method is its dependence on an initialization, X_0. When searching for a stabilizing solution X_s, the initialization X_0 should also be stabilizing, i.e., $(\mathrm{op}(A - BK(X_0)), \mathrm{op}(E))$ should be stable. Except for stable systems, finding a suitable initialization can be a difficult task. Stabilizing algorithms have been proposed, mainly for standard systems, e.g., in (Kleinman, 1968; Varga, 1981; Sima, 1981; Hammarling, 1982). However, often these algorithms produce a matrix X_0 and/or the following several matrices X_i, $i = 1, 2, \ldots$ (computed by the Newton method), with very large norms, and the solver may encounter severe numerical difficulties. For this reason, Newton's method is best used for iterative improvement of a solution or as defect correction method (Mehrmann and Tan, 1988), 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. (Ciubotaru and Staroswiecki, 2009) and the references therein.

The organization of the paper is as follows. Section 2 starts by summarizing the basic theory and Newton's algorithms for AREs. Algorithmic details, computation of the Newton direction, computation of the Newton step size, and convergence tests are discussed in separate subsections. Section 3 presents the main results of an extensive performance investigation of the solvers based on Newton's method, in comparison with the MATLAB solver `care`. Randomly generated systems with order till 1000 (but also a system with order 2000), as well as systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003; Leibfritz and Lipinski, 2004), are considered in the

two subsections. Section 4 summarizes the conclusions.

2 BASIC THEORY AND NEWTON'S ALGORITHMS

The following assumptions are made.

Assumptions A:

- Matrix E is nonsingular.
- Matrix pair $(\mathrm{op}(E)^{-1}\mathrm{op}(A), \mathrm{op}(E)^{-1}B)$ is stabilizable.
- Matrix $R = R^T$ is positive definite ($R > 0$).
- A stabilizing solution X_s exists and it is unique.

The algorithms considered in the sequel are enhancements of Newton's method, which employ a *line search* procedure to minimize the residual along the Newton direction.

The conceptual algorithm can be stated in the following form:

Algorithm N: Newton's method with line search for CARE

Input: The coefficient matrices E, A, B, Q, R, and L, and an initial matrix $X_0 = X_0^T$.
Output: The approximate solution X_k of CARE.

FOR $k = 0, 1, \ldots, k_{\max}$, DO

1. If convergence or non-convergence is detected, return X_k and/or a warning or error indicator value.
2. Compute $K_k := K(X_k)$ with (2) and $\mathrm{op}(A_k)$, where $A_k = \mathrm{op}(A) - BK_k$.
3. Solve in N_k the continuous-time generalized (or standard, if $E = I_n$) Lyapunov equation

$$\mathrm{op}(A_k)^T N_k \mathrm{op}(E) + \mathrm{op}(E)^T N_k \mathrm{op}(A_k) = -\mathcal{R}(X_k).$$

4. Find a step size t_k which minimizes the squared Frobenius norm $\|\mathcal{R}(X_k + tN_k)\|_F^2$ (with respect to t).
5. Update $X_{k+1} = X_k + t_k N_k$.

END

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

Basic properties for the standard and modified Newton's algorithms for CAREs can be stated as follows (Benner, 1997):

Theorem 2.1 (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

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

Theorem 2.2 (Convergence of Algorithm N)**.** *If the Assumptions A hold, X_0 is stabilizing, and, in addition, $(\mathrm{op}(E)^{-1}\mathrm{op}(A), \mathrm{op}(E)^{-1}B)$ is controllable and $t_k \ge t_L > 0$, for all $k \ge 0$, then the iterates of the Algorithm N satisfy*

(a) All iterates X_k are stabilizing.
(b) $\|\mathcal{R}(X_{k+1})\|_F \le \|\mathcal{R}(X_k)\|_F$ and equality holds if and only if $\mathcal{R}(X_k) = 0$.
(c) $\lim_{k\to\infty} \mathcal{R}(X_k) = 0$.
(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 2.2 does not ensure monotonic convergence of the iterates X_k in terms of definiteness, contrary to the standard case (Theorem 2.1, item (b)). On the other hand, under the specified conditions, Theorem 2.2 states the monotonic convergence of the residuals to 0, which is not true for the standard algorithms. It is conjectured that Theorem 2.2 also holds under the weaker assumption of stabilizability instead of controllability. This is supported by the numerical experiments.

2.1 Algorithmic Details

The essential steps of Algorithm N will be detailed below.

Continuous-time AREs can be put in a simpler form, which is more convenient for Newton's algorithms. Specifically, setting

$$\begin{aligned} \tilde{A} &= A - BR^{-1}L^T, \\ \tilde{Q} &= Q - LR^{-1}L^T, \end{aligned} \tag{3}$$

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

$$\begin{aligned} 0 &= \mathrm{op}(A)^T X \mathrm{op}(E) + \mathrm{op}(E)^T X \mathrm{op}(A) \\ &\quad - \mathrm{op}(E)^T X G X \mathrm{op}(E) + Q =: \mathcal{R}(X), \end{aligned} \tag{4}$$

or, in the standard case ($E = I_n$), to

$$0 = \mathrm{op}(A)^T X + X \mathrm{op}(A) - XGX + Q =: \mathcal{R}(X). \tag{5}$$

The transformations in (3) eliminate the matrix L from the formulas to be used. It is more economical to solve the equations (4) or (5), since otherwise the calculations involving L must be performed at each iteration. In this case, the matrix K_k is no longer computed in Step 2, and $A_k = \mathrm{op}(A) - GX_k\,\mathrm{op}(E)$ (or $A_k = \mathrm{op}(A) - DD^TX_k\,\mathrm{op}(E)$).

Algorithm N was implemented in a Fortran 77 subroutine `SG02CD` following the SLICOT Library (Benner et al., 1999; Van Huffel and Sima, 2002; Van Huffel et al., 2004) implementation and documentation standards[1]. The implementation deals with generalized algebraic Riccati equations, possibly for the discrete-time case, without inverting the matrix E. This is very important for numerical reasons, especially when E is ill-conditioned with respect to inversion. Standard algebraic Riccati equations (including the case when E is specified as I_n, or even `[]` in MATLAB), are solved with the maximal possible efficiency. Moreover, both control and filter algebraic Riccati equations 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. It it possible to also avoid the transposition for C and L, for the filter equation, but this is less important and more difficult to implement. (Some existing lower-level routines do not cover the transposed case.)

The implemented algorithm solves either the generalized CARE (4) or standard CARE (5) using Newton's method with or without line search. The selection is made using another option. There is an option for solving related AREs with the minus sign replaced by a plus sign in front of the quadratic term. Moreover, instead of the symmetric matrix G, $G = BR^{-1}B^T$, the n-by-m matrix B and the symmetric and invertible m-by-m matrix R, or its Cholesky factor, may also be given. The iteration is started by an initial (stabilizing) matrix X_0, which can be omitted, if the zero matrix can be used. If X_0 is not stabilizing, and finding X_s is not required, Algorithm N will converge to another solution of CARE. Either the upper, or lower triangles, not both, of the symmetric matrices Q, G (or R), and X_0 need to be stored. 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. In this case, the initial matrix X_0 must 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 method, or the modified Newton method, with line search. The optimal size of the 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.

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.

If $m \leq n/3$, the algorithm is faster if a factorization $G = DD^T$ is used instead of G itself. Usually, the routine uses the Cholesky factorization of the matrix R, $R = L_r^TL_r$, and computes $D = BL_r^{-1}$. The standard theory assumes that R is positive definite. But the routine works also if this assumption does not hold numerically, by using the UDU^T or LDL^T factorization of R. In that case, the current implementation uses G, and not its factors, even if $m \leq n/3$.

The arrays holding the data matrices A and E are unchanged on exit. Array `B` stores either B or G. On exit, if B was given, and $m \leq n/3$, `B` returns the matrix $D = BL_r^{-1}$, if the Cholesky factor L_r can be computed. Otherwise, array `B` is unchanged on exit. Array `Q` stores matrix Q on entry and the computed solution X on exit. If matrix R or its Cholesky factor is given, it is stored in array `R`. On exit, `R` contains either the Cholesky factor, or the factors of the UDU^T or LDL^T factorization of R, if R is found to be numerically indefinite. In that case, the interchanges performed for the UDU^T or LDL^T factorization are stored in an auxiliary integer array.

The basic stopping criterion for the iterative process is stated in terms of a normalized residual, r_k, and a tolerance τ. If

$$r_k := r(X_k) := \|\mathcal{R}(X_k)\|_F / \max(1, \|X_k\|_F) \leq \tau, \quad (6)$$

where X_k is the currently computed approximate solution (at iteration k), 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, for given G, τ is computed by the formula

$$\tau = \min\big(\varepsilon_M\sqrt{n}\big(\|E\|_F\,(2\,\|A\|_F + \|G\|_F\|E\|_F) + \|Q\|_F\big), \sqrt{\varepsilon_M}\big). \quad (7)$$

When G is given in factorized form (see above), then $\|G\|_F$ in (7) is replaced by $\|D\|_F^2$. When E is identity, the factors involving its norm are omitted. The second operand of min in (7) was introduced to prevent deciding convergence too early for systems with very large norms for A, E, G, and/or Q.

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 a working array, where k_s is the iteration number when Newton's process stopped.

Several approaches have been tried in order to reduce the number of iterations. One of them was to

[1] See `http://www.slicot.org`

set $t_k = 1$ whenever $t_k \leq \sqrt{\varepsilon_M}$. Often, but especially 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 finally 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 $\|\hat{\mathcal{R}}(X_k + t_k N_k)\|_F \leq 10$, if this happens during the first 10 iterations; here, $\hat{\mathcal{R}}(X_k + t_k N_k)$ is an estimate of the residual obtained using the formula (10).

In order to observe stagnation, the last computed k_B residuals are stored in the first k_B entries of an array `RES`. If $\|\hat{\mathcal{R}}(X_k + t_k N_k)\|_F > \tau_s \|\mathcal{R}(X_{k-k_B})\|_F > 0$, then $t_k = 1$ is used instead. The current implementation uses $\tau_s = 0.9$ and sets $k_B = 2$, but values as large as $k_B = 10$ 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.

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 G and a copy of the matrix Q, if G 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.

If G is to be used (since $m > n/3$), but the norm of G is too large, then its factor D is used thereafter, in order to enhance the numerical accuracy, even if the efficiency somewhat diminishes.

2.2 Computation of the Newton Direction

The algorithm computes the initial residual matrix $\mathcal{R}(X_0)$ and the matrix $\mathrm{op}(A_0)$, where $A_0 := \mathrm{op}(A) \pm GX_0\,\mathrm{op}(E)$. If no initial matrix X_0 is given, we set $X_0 = 0$, $\mathcal{R}(X_0) = Q$ and $\mathrm{op}(A_0) = A$.

At the beginning of the iteration k, $0 \leq k \leq k_{\max}$, the algorithm decides to terminate or continue the computations, based on the current normalized residual $r(X_k)$. If $r(X_k) > \tau$, a standard (if $E = I_n$) or generalized (otherwise) Lyapunov equation

$$\mathrm{op}(A_k)^T N_k\,\mathrm{op}(E) + \mathrm{op}(E)^T N_k\,\mathrm{op}(A_k) = -\sigma\mathcal{R}(X_k), \tag{8}$$

is solved in N_k (the Newton direction), using SLICOT subroutines. The scalar $\sigma \leq 1$ is set by the Lyapunov solver in order to prevent solution overflowing. Normally, $\sigma = 1$.

Another option is to scale the matrices A_k and E (if E is general) for solving the Lyapunov equations, and suitably update their solutions. Note that the LAPACK subroutines `DGEES` and `DGGES`, (Anderson et al., 1999) which are called by the SLICOT standard and generalized Lyapunov solvers, respectively, to compute the real Schur(-triangular) form, do not scale the cefficient matrices. Just column and row permutations are performed, to separate isolated eigenvalues. For some examples, 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 code.

2.3 Computation of the Newton Step Size

The next step is the computation of the optimal size of the Newton step (line search). The procedure minimizes the Frobenius norm of the residual matrix along the Newton direction, N_k. Specifically, the optimal step size t_k is given by

$$t_k = \arg\min_t \|\mathcal{R}(X_k + tN_k)\|_F^2. \tag{9}$$

It is proved (Benner, 1997) that, in certain standard conditions, an optimal t_k exists, and it is in the "canonical" interval [0,2]. Computationally, t_k is found as the argument of the minimal value in [0,2] of a polynomial of order 4. Indeed,

$$\mathcal{R}(X_k + tN_k) = (1-t)\mathcal{R}(X_k) - t^2 V_k, \tag{10}$$

where $V_k = \mathrm{op}(E)^T N_k G N_k\,\mathrm{op}(E)$. Therefore, the minimization problem (9) reduces to the minimization of the quartic polynomial (Benner, 1997)

$$\begin{aligned} f_k(t) &= \mathrm{trace}(\mathcal{R}(X_k + tN_k)^2) \\ &= \alpha_k(1-t)^2 - 2\beta_k(1-t)t^2 + \gamma_k t^4, \end{aligned} \tag{11}$$

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

In order to solve the minimization problem (9), 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 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). Specifically, let the cubic polynomial be defined by

$$p(t) = a + bt + ct^2 + dt^3.$$

Normally, a matrix pencil is built, whose eigenvalues are the roots of the given polynomial, and they are computed using the QR and QZ algorithms, depending on the magnitude of the polynomial coefficients.

A candidate solution should satisfy the following requirements: (i) it is real; (ii) it is in the interval [0,2]; (iii) the second derivative of the cubic polynomial is positive. 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.

2.4 Convergence Tests and Updating the Current Iterate

The next action is to check if the line search stagnates and/or the standard Newton step is to be preferred. If $n > 1$, $k \leq 10$, $t_k < 0.5$, $\varepsilon_M^{1/4} < r_k < 1$, and $\|\hat{\mathcal{R}}(X_k + t_k N_k)\|_F \leq 10$, or $\|\hat{\mathcal{R}}(X_k + t_k N_k)\|_F > \tau_s \|\mathcal{R}(X_{k-k_B})\|_F$ (i.e., stagnation is detected), then a standard Newton step ($t_k = 1$) is used.

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 improvement can be expected. 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 (10) could suffer from severe numerical cancellation, and hence it could compromise the accuracy of the intermediate results.

Then, $\|X_{k+1}\|_F$ and r_{k+1} are computed, and $k = k+1$ is set. If the chosen step was not a Newton step, but the residual norm increased compared to the previous iteration, i.e., $\|\mathcal{R}(X_{k+1})\|_F \geq \|\mathcal{R}(X_k)\|_F$, but it is less than 1, and the normalized residual is less than $\varepsilon_M^{1/4}$, then the iterative process is terminated and a warning value is set. Otherwise, the iteration continues.

3 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 2011 and MATLAB 8.0.0.783 (R2012b). The SLICOT-based MATLAB executable MEX-functions have been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

3.1 Randomly Generated Systems

A first set of tests refer to CAREs (4) with initial matrices E, A, B, L, Q, and R randomly generated from a uniform distribution in the (0,1) interval, with n and m set as $n = 200:200:1000$, $m = 200:200:n$ (in MATLAB notation). The generated matrix E was stabilized by subtracting 100·norm(E) from the diagonal. The generated matrices Q and R were modified by adding n and m, respectively, to the diagonal entries, and then each of them was symmetrized, by adding its transpose. The generated matrix L was divided by 100. We then used the MATLAB function `care` from the Control System Toolbox (MATLAB, 2011) with inputs A, B, Q, R, L, and E, and stabilized A using $A := A - BF$, where F is the feedback gain matrix returned by `care`. A new Riccati solution was computed by `care` using the modified A and the other matrices. This allowed us to set to zero the initial matrix X_0. For the Newton solver, we removed the effect of L using the formulas (3). Fifteen CARE problems have been generated. For each CARE, various options have been tried (e.g., use either the upper or lower part of symmetric matrices, use the two values of $\mathrm{op}(M)$, use either the matrices B and R, or the matrix G). The default tolerance, computed by the solver when the input value is non-positive, has been used.

Fig. 1 presents the normalized residuals for the random examples solved using Newton solver with line search, and `care`. Fig. 2 presents the CPU times (computed using the MATLAB pair functions `tic` and `toc`). The y-axis is scaled logarithmically, for better clarity, since the CPU times vary significantly. For the largest example, the run time for the Newton solver and $\mathrm{op}(M) = M$ is about half the run time for `care`.

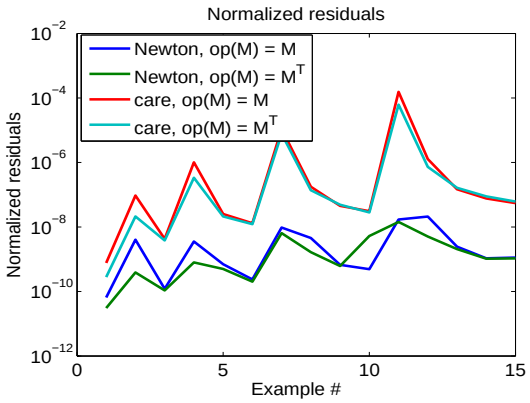

Figure 1: The normalized residuals for random examples using Newton solver with line search and `care`; n = 200 : 200 : 1000, m = 200 : 200 : n.

Similarly, Fig. 3 and Fig. 4 present the normalized

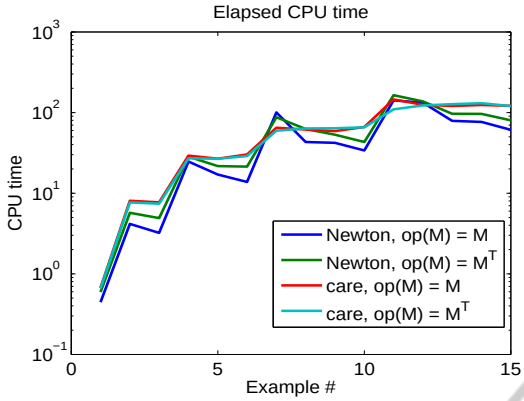

Figure 2: The CPU times for random examples using Newton solver with line search and `care`; n = 200 : 200 : 1000, m = 200 : 200 : n.

residuals and the CPU times, respectively, when using standard Newton solver and `care`. The large error for an example with $n = 600$, $m = 200$ (and $\mathrm{op}(M) = M^T$) is not typical.

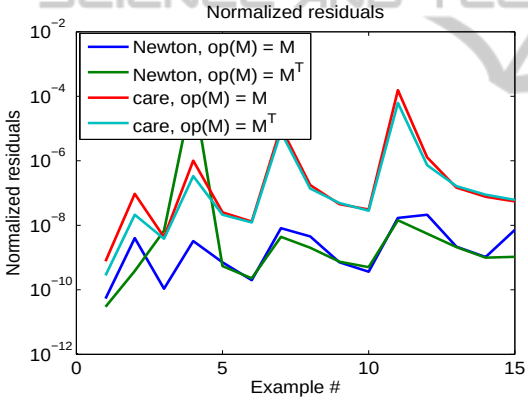

Figure 3: The normalized residuals for random examples using standard Newton solver and `care`; n = 200 : 200 : 1000, m = 200 : 200 : n.

For both variants, the Newton solver was almost always faster than `care`. It was also (with one exception) significantly more accurate. Note that for this set of tests, the problems with $\mathrm{op}(M) = M^T$ needed more iterations and CPU time for Newton solver than those with $\mathrm{op}(M) = M$, especially for the standard Newton solver. Indeed, the standard Newton solver was most often over 50% faster than `care` for $\mathrm{op}(M) = M$, but often over 20% slower than `care` for $\mathrm{op}(M) = M^T$.

The Euclidean norm of the vectors of normalized residuals (one normalized residual for each example) and the mean number of iterations are shown in Table 1 for the case $\mathrm{op}(M) = M$.

We have also solved a problem with $n = m = 2000$, built as above. Newton solver with line search needed 4 iterations when $\mathrm{op}(M) = M$, and 7 iterations when $\mathrm{op}(M) = M^T$. The CPU times were about 793 and 1360 seconds, and the normalized residuals were $8.41 \cdot 10^{-9}$ and $4.4 \cdot 10^{-9}$, respectively. MATLAB `care` needed about 1350 and 1530 seconds, and the normalized residuals were $2.31 \cdot 10^{-7}$ and $3.66 \cdot 10^{-7}$, respectively. Similarly, when $E = I_n$, the results for the Newton solver were: 4 and 9 iterations, 68 and 469 seconds, and normalized residuals $1.69 \cdot 10^{-10}$ and $3.22 \cdot 10^{-13}$, respectively. MATLAB `care` needed 99.4 and 100 seconds, and the normalized residuals were $1.61 \cdot 10^{-11}$ and $1.01 \cdot 10^{-11}$, respectively.

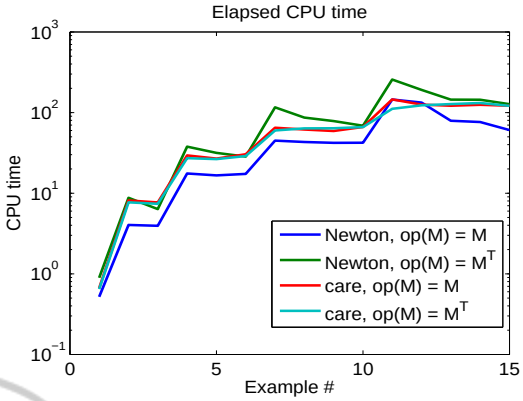

Figure 4: The CPU times for random examples using standard Newton solver and `care`; n = 200 : 200 : 1000, m = 200 : 200 : n.

Table 1: Normalized residuals 2-norms and mean number of iterations for random examples.

	L. search	Standard	`care`
$\|r_{1:15}\|_2$	$2.98 \cdot 10^{-8}$	$3.01 \cdot 10^{-8}$	$1.55 \cdot 10^{-4}$
$\frac{1}{15}\sum_1^{15} k_s^i$	5.6	5.33	–

3.2 Systems from the COMPl$_e$ib Collection

Other tests have been performed for linear systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003; Leibfritz and Lipinski, 2004). This collection contains 124 standard continuous-time examples (with $E = I_n$), with several variations, giving a total of 168 problems. All but 16 problems (for systems of order larger than 2000, with matrices in sparse format) have been tried. The performance index matrices Q and R have been chosen as identity matrices of suitable sizes. The matrix L was always zero. Most often we used the default tolerance.

In a series of tests, we used X_0 set to a zero matrix, if A is stable; otherwise, we tried to initialize the Newton solver with a matrix computed using the algorithm in (Hammarling, 1982), and when this algo-

rithm failed to deliver a stabilizing initialization, we used the solution provided by the MATLAB function `care`. A zero initialization was used for 44 stable examples. Stabilization algorithm was tried on 107 unstable systems, and succeeded for 91 examples. Failures occurred for 16 examples. With default tolerance, the implementation of the Newton solver used in the preliminary version of this paper did not improve the `care` solution, returning with 0 iterations. But, modifying the test at Step 1 of Algorithm N, in order to continue the calculations at iteration $k = 0$, enabled to improve the accuracy of `care` solution for 15 examples. (Only the solution for example ROC5 could not be improved.) The function `care` failed to solve the Riccati equation for example REA4, with the error message "There is no finite stabilizing solution". This unstable example has been excluded from our tests, because it could not be stabilized.

We tried both standard and modified Newton's method, with or without balancing the coefficient matrices of the Lyapunov equations. The modified solver needed more iterations than the standard solver for 10 examples only. The cumulative number of iterations with modified and standard solver for all 150 examples was 1654 and 2289, respectively. With balancing, the total number of iterations was 1657 and 2279, respectively. The mean number of iterations was about 11, for the modified solver, and 15.2, for the standard solver. We tried also to use the stabilization algorithm whenever possible, including for stable A matrices. Doing so, the total number of iterations without balancing was 1796 and 2208, respectively (1784 and 2207, with balancing).

Fig. 5 shows the normalized residuals for the COMPl$_e$ib examples. For clarity, only the results for Newton solver with line search without balancing and for `care` are plotted. Note that the normalized residual is higher than 1 for the TL example when using `care`. (Its value is $2.13 \cdot 10^3$ for `care`, but $1.09 \cdot 10^{-3}$ for the Newton solver, and $1.32 \cdot 10^{-3}$, using a stabilizing $X_0 \neq 0$.) 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}). Omitting example TL, the maximum normalized residual was of order 10^{-6} for the standard Newton solver, and of order 10^{-9} (10^{-10} with balancing) for the modified solver and `care`.

Similarly, Fig. 6 shows the relative residuals, computed in a similar manner with that used in `care`. The maximum value of these residuals is $8.98 \cdot 10^{-9}$ for the modified Newton solver (for example ROC5), and $3.16 \cdot 10^{-5}$ for `care` (for example TL) . (Its value was 1 for the standard Newton solver and example TL!) Omitting example TL, the maximum relative residual

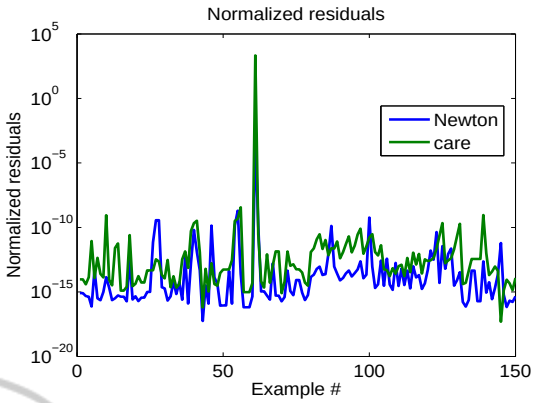

Figure 5: The normalized residuals for examples from the COMPleib collection, using Newton solver with line search without balancing and `care`.

was of order 10^{-7} for the standard Newton solver and of order 10^{-6} for `care`.

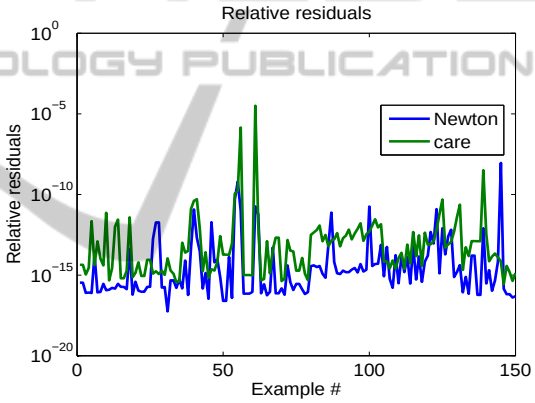

Figure 6: The relative residuals for examples from the COMPleib collection, using Newton solver with line search and `care`.

Figure 7 shows the number of iterations of the Newton solver with line search for the COMPl$_e$ib examples. The largest number, 34, was applied for example CM5_IS, with order $n = 480$, and $m = 1$.

Similarly, Fig. 8 shows the elapsed CPU times. Although the modified Newton method was faster than `care` for 100 examples, out of 150, the sum of the CPU times was about 64% larger than for `care`. This is mainly due to the fact that, with the chosen initialization, some large examples (mainly, 15 examples in the HF2D class) required at least 19 iterations. The standard Newton solver was globally over 25% slower than the solver with line search. The balancing option increased the CPU times by less than 4% in both cases. When using stabilizing $X_0 \neq 0$, the speed-up of the modified Newton solver increased by about 30%; the main contribution came from solving the ARE for example CM6 ($n = 960$, $m = 1$) in just

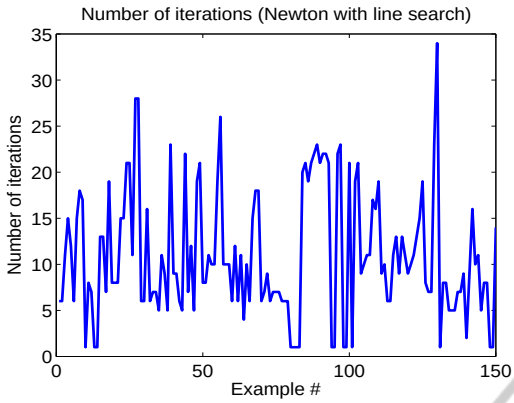

Figure 7: The number of iterations performed by the Newton solver with line search for examples from the COMPleib collection.

one iteration, compared to 19 iterations needed when $X_0 = 0$ was used. (Note that the stabilization algorithm did not work for example CM6, so X_0 was set to the `care` solution.) Clearly, a good initialization could significantly reduce the number of iterations.

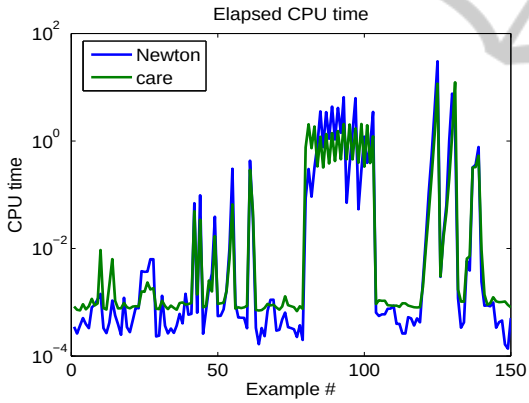

Figure 8: The elapsed CPU time needed by the Newton solver with line search and MATLAB `care` for examples from the COMPleib collection.

When the solution returned by `care` was used to initialize the Newton solver variants for all COMPl$_e$ib examples, with default tolerance and without the modification, mentioned before, of the test at Step 1, the total number of iterations was 69, namely 50 iterations for TL, and one iteration for other 19 examples. The remaining examples were solved without any iterations, since `care` results were accurate enough with the default tolerance. The sum of the CPU times for Newton solver was about 2.4 seconds, compared to about 53.8 seconds for `care`. The normalized residual decreased to $1.05 \cdot 10^{-3}$ for example TL, but the relative residual slightly increased to $9.76 \cdot 10^{-6}$. Omitting the example TL, the maximum normalized residual decreased to $9.17 \cdot 10^{-10}$ (for example EB6, with $n = 160$, $m = 1$), and the maximum relative residual was $3.85 \cdot 10^{-10}$ (for example ISS2, with $n = 270$, $m = 3$).

After modifying the test at Step 1 to force the solver to try at least an update, and after adding a test of relative residual, the total number of iterations increased to 159, namely 11 iterations for example TL, zero iterations for ROC5, and one iterations for the other examples. The sum of the CPU times for Newton solver was then about 8.3 seconds. The normalized residual for example TL varied in between $9.3 \cdot 10^{-4}$ and $1.3 \cdot 10^{-3}$ for the four variants (with/without line search and with/without balancing), and the relative residual varied between $1.61 \cdot 10^{-11}$ and $2.05 \cdot 10^{-11}$. Omitting the example TL, the maximum normalized residual decreased to $6.9 \cdot 10^{-13}$ for Newton solver, and $3.6 \cdot 10^{-9}$ for `care` (for example HF2D_IS5, with $n = 5$, $m = 2$), and the maximum relative residual was $8.42 \cdot 10^{-13}$ (for example CBM, with $n = 348$, $m = 1$). The decision to keep this modification of the tests was based on these improved results.

We used the same initialization provided by `care` with values for the tolerance parameter τ set to 10^{-12}, 10^{-14}, and relative machine precision, ε_M.

For $\tau = 10^{-12}$, the behavior was identical with that for the default τ. For $\tau = 10^{-14}$, example TL needed 50 iterations without convergence (the tolerance being too small), one example needed 6 iterations, 2 examples needed 5 iterations, 3 examples needed 4 iterations, 14 examples needed 3 iterations, 9 examples needed 2 iterations, 119 examples needed one iteration, and ROC5 needed no iteration. The repartition of the number of iterations for $\tau = \varepsilon_M$ is shown in the bar graph from Fig. 9, where TL example was excluded, for clarity. Only for ROC5 example, the Newton solver returned before finishing the first iteration (reporting 0 iterations); it found that no improvement of X_0 is numerically possible, since the norm of the correction $t_0 N_0$, $3.07 \cdot 10^{-14}$, was too small compared to the norm of X_0, which is $1.41 \cdot 10^{4}$. (The normalized residual value for X_0 was $4.88 \cdot 10^{-18}$.) Omitting TL, the maximum normalized residual reduced to $6.9 \cdot 10^{-13}$ for $\tau = 10^{-12}$, and to about $3.5 \cdot 10^{-13}$ for smaller values of τ. Performance results are summarized in Table 2.

Table 2: Summary of performance results for small tolerance τ and initialization by MATLAB `care`.

τ	Total iterations	Sum of CPU times
10^{-12}	198	10.83
10^{-14}	257	23.79
ε_M	344	26.23

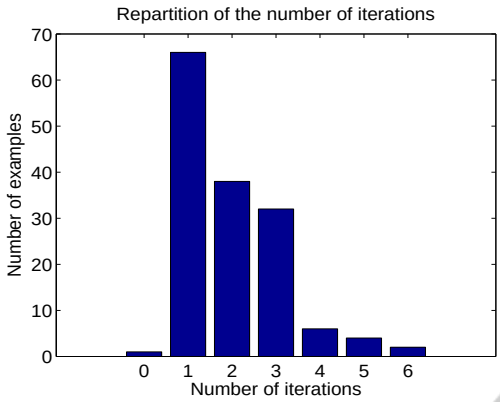

Figure 9: Bar graph showing the repartition of the number of iterations for examples from the COMPleib collection, using Newton solver with line search without balancing, initialization using `care` and tolerance relative machine precision.

Fig. 10 shows the normalized residuals for the Newton solver with line search, initialized by `care`, and with tolerance $\tau = \varepsilon_M$. Clearly, Newton solver reduces the residuals by several orders of magnitude, compared to `care`.

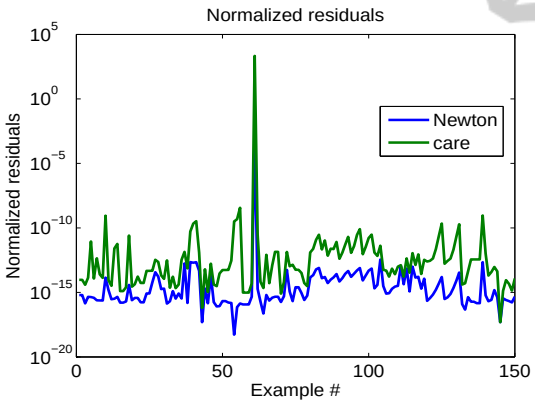

Figure 10: The normalized residuals for examples from the COMPleib collection, using Newton solver with line search without balancing, initialization using `care` and tolerance relative machine precision.

Fig. 11 shows similarly the relative residuals. Except for ROC5, Newton solver always reduces the residuals, often by several orders of magnitude, compared to `care`. Fig. 12 shows by a bar graph the size of this improvement. Specifically, the improvement is of seven orders of magnitude for one example, six orders for three examples, four orders for five examples, etc. For 114 examples, the improvement is between one and three (inclusive) orders of magnitude.

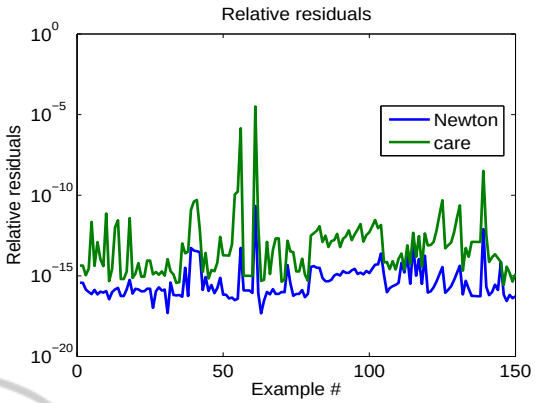

Figure 11: The relative residuals for examples from the COMPleib collection, using Newton solver with line search without balancing, initialization using `care` and tolerance relative machine precision.

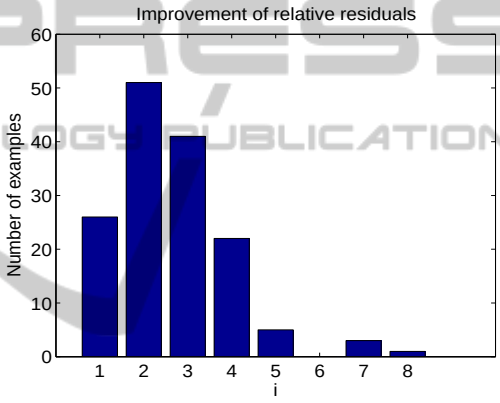

Figure 12: Bar graph showing the improvement of the relative residuals for examples from the COMPleib collection, using Newton solver with line search without balancing, initialization using `care` and tolerance relative machine precision. The height of the i-th vertical bar indicates the number of examples for which the improvement was between i-1 and i orders of magnitude.

4 CONCLUSIONS

Basic theory and improved algorithms for solving continuous-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 (finding the Newton direction, finding the Newton step size), 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 using the widely-used MATLAB function `care`. Randomly generated systems with orders till 1000 (and even a system with order 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.

ACKNOWLEDGEMENTS

Part of this work was done many years ago in a research stay at the Technical University Chemnitz, Germany, during November 1 – December 20, 2005, with the financial support from the German Science Foundation. The long cooperation with Peter Benner from Technical University Chemnitz and Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany, is much acknowledged. Thanks are also addressed to Martin Slowik from Institut für Mathematik, Technical University Berlin, who worked out (till 2005) a preliminary version of the SLICOT codes for continuous-time algebraic Riccati equations. The work has been recently resumed by the author. Finally, the continuing support from the NICONET e.V. is warmly acknowledged.

REFERENCES

Abels, J. and Benner, P. (1999). CAREX—A collection of benchmark examples for continuous-time algebraic Riccati equations (Version 2.0). SLICOT Working Note 1999-14, Katholieke Universiteit Leuven, ESAT/SISTA, Leuven, Belgium. Available from http://www.slicot.org.

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.

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

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*, pages 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., Byers, R., Losse, P., Mehrmann, V., and Xu, H. (2007). Numerical solution of real skew-Hamiltonian/Hamiltonian eigenproblems. Technical report, Technische Universität Chemnitz, Chemnitz.

Benner, P., Byers, R., Mehrmann, V., and Xu, H. (2002). Numerical computation of deflating subspaces of skew Hamiltonian/Hamiltonian pencils. *SIAM J. Matrix Anal. Appl.*, 24(1):165–190.

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*, volume 1, chapter 10, pages 499–539. Birkhäuser, Boston.

Benner, P. and Sima, V. (2003). Solving algebraic Riccati equations with SLICOT. In *CD-ROM Proceedings of The 11th Mediterranean Conference on Control and Automation MED'03, June 18–20 2003, Rhodes, Greece*. Invited session IV01, "Computational Toolboxes in Control Design", Paper IV01-01, 6 pages.

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

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

Francis, B. A. (1987). *A Course in H_∞ Control Theory*, volume 88 of *Lect. Notes in Control and Information Sciences*. Springer-Verlag, New York.

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.

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.

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

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

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–50(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. (2003). Description of the benchmark examples in *COMPl$_e$ib*. Technical report, Department of Mathematics, University of Trier, D–54286 Trier, Germany.

Leibfritz, F. and Lipinski, W. (2004). *COMPl$_e$ib* 1.0 – User manual and quick reference. Technical report, Department of Mathematics, University of Trier, D–54286 Trier, Germany.

MATLAB (2011). Control System Toolbox User's Guide. Version 9.

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.

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

Penzl, T. (2000). LYAPACK Users Guide. Technical Report SFB393/00–33, Technische Universität Chemnitz, Sonderforschungsbereich 393, "Numerische Simulation auf massiv parallelen Rechnern", Chemnitz.

Raines, III, A. C. and Watkins, D. S. (1992). A class of Hamiltonian-symplectic methods for solving the algebraic Riccati equation. Technical report, Washington State University, Pullman, WA.

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. (1981). An efficient Schur method to solve the stabilizing problem. *IEEE Trans. Automat. Contr.*, AC–26(3):724–725.

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

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

Sima, V. (2010). Structure-preserving computation of stable deflating subspaces. In Kayacan, E., editor, *Proceedings of the 10th IFAC Workshop "Adaptation and Learning in Control and Signal Processing" (ALCOSP 2010), Antalya, Turkey, 26–28 August 2010 (CD-ROM)*, 6 pages. IFAC-PapersOnLine, Volume 10, Part 1, http://www.ifac-papersonline.net/Detailed/46793.html.

Sima, V. (2011). Computational experience with structure-preserving Hamiltonian solvers in optimal control. In Ferrier, J.-L., Bernard, A., Gusikhin, O., and Madani, K., editors, *Proceedings of the "8th International Conference on Informatics in Control, Automation and Robotics" (ICINCO 2011), Noordwijkerhout, The Netherlands, 28–31 July, 2011 (CD-ROM)*, volume 1, pages 91–96. SciTePress—Science and Technology Publications.

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 (CD-ROM)*. Omnipress. Session FEA2: *Control Systems 5*, Paper FEA2-3.

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), Hilton Palacio del Rio Hotel, San Antonio, Texas, U.S.A., September 3–5, 2008*, pages 601–606. Omnipress.

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

Van Huffel, S. and Sima, V. (2002). SLICOT and control systems numerical software packages. In *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, Scottish Exhibition and Conference Centre, Glasgow, Scotland, U.K.*, pages 39–44. Omnipress.

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.

Varga, A. (1981b). A Schur method for pole assignment. *IEEE Trans. Automat. Contr.*, AC–26(2):517–519.