SOLVING DISCRETE-TIME ALGEBRAIC RICCATI EQUATIONS USING MODIFIED NEWTON'S METHOD

Vasile Sima
Advanced Research
National Institute for Research & Development in Informatics
Romania
vsima@ici.ro

Abstract

Improved algorithms for solving discrete-time algebraic Riccati equations using Newton's method with or without line search are described. The numerical results illustrate the advantages such algorithms offer, especially for improving the solutions computed by other solvers.

Key words

Algebraic Riccati Equation; Numerical Algorithms; Numerical Methods; Optimal Control; Optimal Estimation.

1 Introduction

The numerical solution of algebraic Riccati equations (AREs) is required in many computational algorithms for linear systems control, filtering, model reduction, etc. 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. The generalized discrete-time AREs (DAREs), with unknown $X = X^T \in \mathbf{R}^{n\times n}$, are defined by

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

where the operator $\mathrm{op}(M)$ represents either M or M^T, E and $\hat{R}(X)$ are assumed nonsingular, and

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

with L of suitable size. Define also $G := B\hat{R}(X)^{-1}B^T$. Note that G depends on X (contrary to the continuos-time case, when $\hat{R}(X)$ is replaced by R), but this dependency was suppressed, for convenience. An optimal regulator problem involves the solution of a DARE with $\mathrm{op}(M) = M$; an optimal estimator problem involves the solution of a DARE 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. In practice, often Q and L are given as $C^T\bar{Q}C$ and $L = C^T\bar{L}$, respectively. 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 $(\,\mathrm{op}(A - BK(X_s))\,,\,\mathrm{op}(E)\,)$ 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

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

with X replaced by X_s.

Newton's method for solving AREs has been considered by many authors, for instance, (Benner and Byers, 1998; Kleinman, 1968). 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), etc.

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 is preferred in implementing certain fault-tolerant systems, which require controller updating.

2 Basic Theory and Newton's Algorithms

The following assumptions are made.

Assumptions A:

1. Matrix E is nonsingular.
2. Matrix pair $(\,\mathrm{op}(E)^{-1}\,\mathrm{op}(A)\,,\,\mathrm{op}(E)^{-1}B)$ 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 $\hat{R}(X_s) > 0$.

The algorithms considered in the sequel are enhancements 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:

Algorithm ND: Modified Newton method for DARE

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 ARE.

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 (3) and $\mathrm{op}(A_k)$, where $A_k = \mathrm{op}(A) - BK_k$.
3. Solve in N_k the discrete-time generalized (or standard, if $E = I_n$) Stein equation (also called discrete-time Lyapunov equation)

$$\mathrm{op}(A_k)^T N_k \,\mathrm{op}(A_k) - \mathrm{op}(E)^T N_k \,\mathrm{op}(E) = -\mathcal{R}(X_k)\,. \quad (4)$$

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

The standard Newton's algorithm is 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 DAREs are stated, e.g., in (Benner, 1997), for instance:

Theorem 2.1 (Convergence for standard case). *If Assumptions A hold, and X_0 is stabilizing, then the iterates of the Algorithm ND with $t_k = 1$ satisfy*

(a) All matrices X_k are stabilizing.
(b) $X_s \le \cdots \le X_{k+1} \le X_k \le \cdots \le X_1$.
(c) $\lim_{k\to\infty} X_k = X_s$.
(d) Global quadratic convergence: There is a constant $\gamma > 0$ such that

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

For continuous-time AREs (CAREs), a similar convergence theorem is stated for the modified Newton's algorithm. The corresponding theorem does not ensure monotonic convergence of the iterates X_k in terms of definiteness, contrary to the standard case (Theorem 2.1, (b)). On the other hand, under mild conditions, it states the monotonic convergence of the residuals to 0, which is not true for the standard algorithm.

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

2.1 Algorithmic Details

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

$$\tilde{A} = A - BR^{-1}L^T, \quad \tilde{Q} = Q - LR^{-1}L^T, \quad (6)$$

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

$$0 = \mathrm{op}(A)^T X \,\mathrm{op}(A) - \mathrm{op}(E)^T X \,\mathrm{op}(E) - \mathrm{op}(A)^T XGX \,\mathrm{op}(A) + Q =: \mathcal{R}(X), \quad (7)$$

where the second term reduces to X in the standard case ($E = I_n$). The transformations in (6) eliminate the matrix L from the formulas to be used. In this case, the matrix K_k is no longer computed in Step 2, and $A_k = \mathrm{op}(A) - GX_k \,\mathrm{op}(A)$ (or $A_k = \mathrm{op}(A) - DD^T X_k \,\mathrm{op}(A)$, where $G =: DD^T$).

Algorithm ND was implemented in a Fortran 77 subroutine, `SG02CD`, following the SLICOT Library (Benner, Mehrmann, Sima, Van Huffel and Varga, 1999) implementation and documentation standards (see `http://www.slicot.org`). The same routine also solves CAREs. The implementation deals with generalized algebraic Riccati equations without inverting the matrix E, which is very important for numerical reasons, since E might be ill-conditioned with respect to inversion. Standard algebraic Riccati equations (including the case when E is set to 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 transposing C, for the filter equation, but this is less important and more difficult to implement at the SLICOT Library level, since some existing lower-level routines do not directly cover the transposed case. But this issue was solved in the executable MEX-file calling the SLICOT Riccati solver.

The implemented algorithm solves either the generalized or standard DARE (7) using Newton's method with or without line search. There is an option for solving related AREs with the minus sign replaced by a plus sign in front of the quadratic term. While G is sometimes used in the code for solving DAREs (see below), it is not allowed to give it on input, instead of the matrices B and R, contrary to the CAREs case. 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 ND will converge to another solution of DARE. Either the upper, or lower triangles, not

both, of the symmetric matrices Q, G, 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 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.

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 $\hat{R}(X_k)$, $\hat{R}(X_k) = L_r^T L_r$, and computes $D = BL_r^{-1}$. The standard theory assumes that $\hat{R}(X_s)$ is positive definite. But the routine works also if this property does not hold numerically for $\hat{R}(X_k)$, by using the UDU^T or LDL^T factorization of $\hat{R}(X_k)$. In that case, the 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 `Q` stores matrix Q on entry and the computed solution X on exit. The array `B`, storing B on input, returns the final matrix $D = BL_r^{-1}$, if $m \leq n/3$ and the Cholesky factor L_r can be computed. Similarly, the array `R`, storing R on input, may return either the Cholesky factor, 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 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

$$r_k := \|\mathcal{R}(X_k)\|_F / \max(1, \|X_k\|_F) \leq \tau, \qquad (8)$$

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

$$\tau = \min\big(\, \varepsilon_M \sqrt{n}\big(\|A\|_F\,(\|A\|_F + \|G\|_F\|A\|_F + \|E\|_F^2) + \|Q\|_F\big), \sqrt{\varepsilon_M}\,\big). \qquad (9)$$

The second operand of min in (9) 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 the working array.

For systems with very large norms of the matrices A, E, G, or Q, and small norm of the solution X, the termination criterion involving (9) 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 relative residual, which includes the norms of A, E, G, and Q 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 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 (see (11) below) 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 $\|\hat{\mathcal{R}}(X_k + t_k N_k)\|_F \leq 10$, 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 (11).

In order to detect stagnation, the last computed k_B residuals are stored in 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 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.

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.

2.2 Computation of the Newton direction and step size

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

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 possibly relative residual $r_r(X_k)$). If $\min(r_k, r_r(X_k)) > \tau$, a standard (if $E = I_n$) or generalized Stein equation (4) is solved in N_k (the Newton direction), using SLICOT subroutines.

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 (`www.netlib.org/lapack/`) subroutines `DGEES` and `DGGES`, which are called by the SLICOT standard and generalized Stein 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, and no scaling, the convergence was not achieved in a reasonable number of iterations.

While for CAREs it is possible to find the optimal step size, by minimizing the Frobenius norm of the residual matrix along the Newton direction, N_k, this is computationally not attractive for DAREs. Specifically, the optimal step size t_k is given by

$$t_k = \arg\min_t \|\mathcal{R}(X_k + tN_k)\|_F^2. \qquad (10)$$

For DARE, t_k is found numerically as the argument of the minimal value in [0,2] of a polynomial of order 4 (Benner, 1997). Indeed,

$$\mathcal{R}(X_k + tN_k) = (1-t)\mathcal{R}(X_k) - t^2 V_k, \qquad (11)$$

where $V_k = \text{op}(A)^T N_k G_k N_k\,\text{op}(A)$ with $G_k := B\hat{R}(X_k)^{-1}B^T$. Therefore, the minimization problem (10) reduces to the minimization of the approximate quartic polynomial (Benner, 1997)

$$\begin{aligned} f_k(t) &= \text{trace}(\mathcal{R}(X_k + tN_k)^2) \\ &= \alpha_k(1-t)^2 - 2\beta_k(1-t)t^2 + \gamma_k t^4, \end{aligned} \qquad (12)$$

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

In order to solve the minimization problem (10), 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, for DAREs, the true $f_k(t)$ is not a polynomial, but 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 $|t| < 1/\|G_k N_k\|$, 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 $\|G_k N_k\| < 1/2$. It can be shown (Benner, 1997) that if X_k is stabilizing, then either N_k is a descent direction for $\|\mathcal{R}(X)\|_F^2$, or $X_k = X_s$. But the stabilizability property is not guaranteed, at least for $t \in [0, 2]$. When $\|G_k N_k\|$ is large (usually, at the beginning of the iterative Newton's process), the step sizes t_k could be too small, and the progress of the iteration could be too slow. A backtracking approach, proposed in (Benner, 1997), can be used to increase the speed of the iterative process. Specifically, in a *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, 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.

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 $\epsilon_M \approx 2.22 \times 10^{-16}$, using Windows 7 Professional (Service Pack 1) operating system (64 bit), Intel Visual Fortran Composer XE 2011 and MATLAB 8.0.0.783 (R2012b). A SLICOT-based MATLAB executable MEX-function has been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

A first set of tests refer to DAREs (7) with initial matrices E, A, B, 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 then modified by subtracting $100 \cdot \text{norm}(E)$ 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 matrix L was set to zero. (Randomly generated L have also been tried.) We then used the MATLAB function `dare` 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 `dare`. A new Riccati solution was computed by `dare` using the modified A and the other matrices. This allowed

us to set to zero the initial matrix X_0. Fifteen DARE problems have been generated. For each DARE, various options have been tried (e.g., use either the upper or lower part of symmetric matrices, or use the two values of op(M)). The default tolerance has been used.

Fig. 1 presents the normalized residuals for the random examples solved using Newton solver with line search, and `dare`. 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.

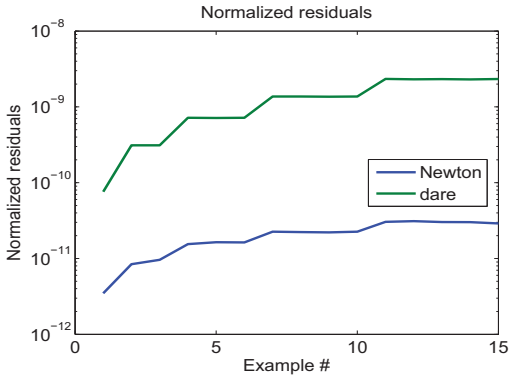

Figure 1. The normalized residuals for random examples using Newton solver with line search and `dare`; $n = 200 : 200 : 1000, m = 200 : 200 : n$.

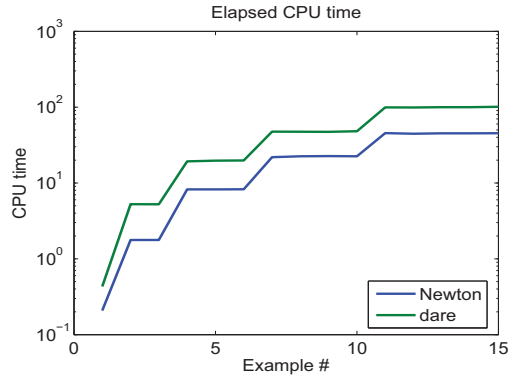

Figure 2. The CPU times for random examples using Newton solver with line search and `dare`; $n = 200 : 200 : 1000$, $m = 200 : 200 : n$.

Similar results have been obtained using standard Newton solver. For both variants, the new solver was always faster and more accurate than `dare`. Note that for this set of tests, the problems with op(M) $= M^T$ and $E = I_n$ usually needed more iterations and CPU time for Newton solver than those with op(M) $= M$.

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, both for E general and $E = I_n$ (in the upper and lower part, respectively). Using the hybrid strategy, the same values as for the standard solver have been obtained.

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

E general	L. search	Standard	`dare`
$\|r_{1:15}\|_2$	$8.7 \cdot 10^{-11}$	$8.6 \cdot 10^{-11}$	$6{\cdot}10^{-9}$
$\frac{1}{15}\sum_1^{15} k_s^i$	2	2	–
$E = I_n$	L. search	Standard	`dare`
$\|r_{1:15}\|_2$	$4.1 \cdot 10^{-9}$	$5.9 \cdot 10^{-9}$	$1.3{\cdot}10^{-5}$
$\frac{1}{15}\sum_1^{15} k_s^i$	20.4	3.7	–

Other tests have been performed for linear systems from the COMPl$_e$ib collection (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. 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 L 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 didn't satisfy the needed conditions, and couldn't be solved by the MATLAB function `dare`, which gave the error message "There is no finite stabilizing solution". These examples have been omitted. 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 (Armstrong and Rublein, 1976), and when this algorithm failed to deliver a stabilizing initialization, we used the solution provided by `dare`. A zero initialization could be used for 6 stable examples. Stabilization algorithm was tried on 82 unstable systems, and succeeded for 55 examples. Failures occurred for 27 examples. Both standard and modified Newton's method, with or without balancing the coefficient matrices of the Stein equations were tried.

Due to paper length restrictions, only the elapsed CPU times are shown, in Fig. 3. The ratio of the sum of the CPU times for `dare` and the modified Newton solver was about 1.95 (and 2.5 for the hybrid strategy). The standard Newton solver was even faster, the ratio being 3.6.

Clearly, a good initialization could improve the behavior of the Newton solver. When the solution returned by `dare` was used for all COMPl$_e$ib examples, and with the tolerance set to ε_M, the new solver still not improved the `dare` results for 5 examples. Alternative

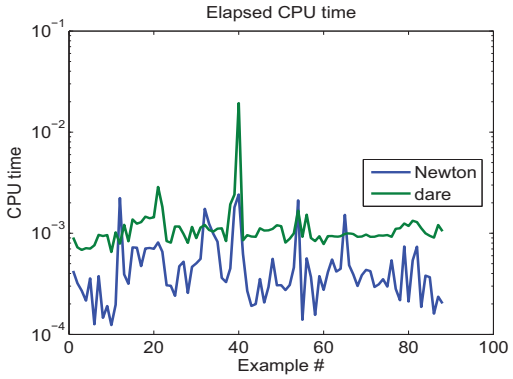

Figure 3. The elapsed CPU time needed by the Newton solver with line search and MATLAB `dare` for examples from the COMPl$_e$ib collection.

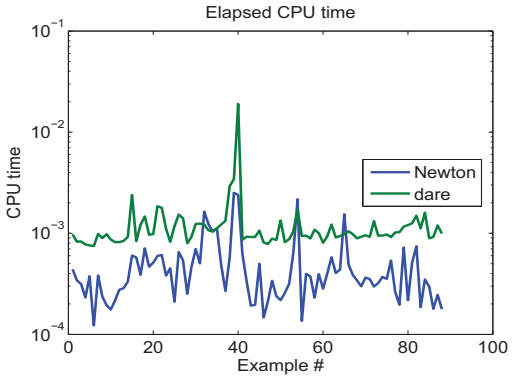

Figure 5. The elapsed CPU time needed by the Newton solver with line search, initialized by `dare` solution, and with tolerance ϵ_M, for examples from the COMPl$_e$ib collection. The elapsed CPU time needed by the MATLAB `dare` is also shown.

algorithms, including evolutionary ones (e.g., (Zelinka et al., 2010)), might be tried for initialization.

Fig. 4 shows the relative residuals for Newton solver with line search, initialized by `dare`, and with tolerance $\tau = \epsilon_M$. Newton solver often significantly reduces the residuals, compared to `dare`. Five examples need further investigation. Similarly, Fig. 5 shows the CPU times of the Newton solver with line search.

4 Conclusion

Basic theory and improved algorithms for solving discrete-time algebraic Riccati equations using Newton's method with or without line search have been presented. The numerical results for discrete-time AREs illustrate the advantages of using such algorithms.

Acknowledgements

The cooperation with Peter Benner and the NICONET support are much acknowledged.

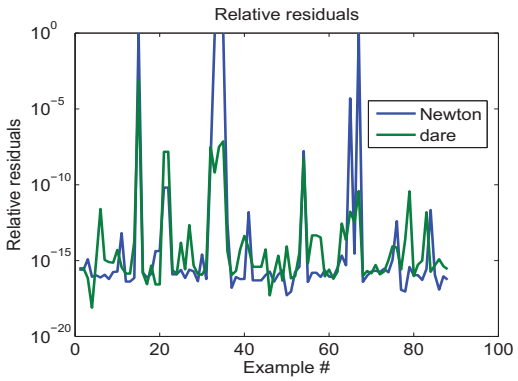

Figure 4. The relative residuals for examples from the COMPl$_e$ib collection, using Newton solver with line search, initialized by `dare` solution and with tolerance ϵ_M. The relative residuals for the MATLAB `dare` are also shown.

References

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

Benner, P. (1997). Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems. Dissertation, Fak. für Mathematik, Techn. Universität Chemnitz–Zwickau.

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

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), pp. 101–107.

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

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

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

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

MathWorks (2011). *Control System Toolbox User's Guide. Version 9*. The MathWorks, Inc.

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

Zelinka, I., Celikovsky, S., Richter, H. and Chen, G., eds (2010). *Evolutionary Algorithms and Chaotic Systems*. Springer.