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

Vasile Sima[1] and Peter Benner[2]
[1]*Advanced Research, 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, 39106 Magdeburg, Germany*

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

Abstract: Refined algorithms for solving continuous-time algebraic Riccati equations using Newton's method with or without line search are discussed. Their main properties are briefly presented. Algorithmic details incorporated in the developed solver are described. The results of an extensive performance investigation on a large collection of examples are summarized. Several numerical difficulties and observed unexpected behavior are reported. These algorithms are strongly recommended for improving the solutions computed by other solvers.

1 INTRODUCTION

Let A, $E \in \mathbf{R}^{n\times n}$, B, $L \in \mathbf{R}^{n\times m}$, Q and R be symmetric, $Q = Q^T \in \mathbf{R}^{n\times n}$, $R = R^T \in \mathbf{R}^{m\times m}$, and E and R assumed nonsingular. The generalized continuous-time algebraic Riccati equations (AREs), or CAREs, with unknown $X = X^T \in \mathbf{R}^{n\times n}$, are defined by

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

with

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

The operator $\mathrm{op}(M)$ represents either M or M^T. Define also $G := BR^{-1}B^T$. The numerical solution of algebraic Riccati equations is required in many computational methods for model reduction, filtering, and controller design for linear control systems (Lancaster and Rodman, 1995; Sima, 1996; Bini et al., 2012). 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 the transpose of the *output matrix* $C \in \mathbf{R}^{p\times n}$, and m replaced by p. 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 (i.e., all eigenvalues have strictly negative real parts), 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).

The literature concerning AREs and their use is overwhelming; see, e.g., the monographs (Anderson and Moore, 1971; Mehrmann, 1991; Lancaster and Rodman, 1995; Bini et al., 2012) for many theoretical results. Numerous numerical methods have been proposed, and several often-used software implementations are available, e.g., in MATLAB (MATLAB, 2011), or in the SLICOT Library.[1] Newton's method for solving AREs, often attributed to Kleinman, has been considered by many authors, see, e.g., (Kleinman, 1968; Mehrmann, 1991; Lancaster and Rodman, 1995; Sima, 1996; Benner and Byers, 1998) and the references therein. Despite of all these efforts, our opinion is that its implementation may still be improved. This involves deep investigation and comparative analysis of numerical properties and performance on large collections of relevant and significant examples of systems. This paper intends to follow such an endeavor, based on the results obtained on the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003). While good performance is already offered by some software tools, this investigation revealed that further improvements might still be possible. Moreover, examples were found for which the expected behavior has not been observed. These examples have been further analyzed to find the underlying reasons, and cures for improving the behavior.

The paper makes use of a refined Newton solver

[1]www.slicot.org

Sima V. and Benner P..
Numerical Investigation of Newton's Method for Solving Continuous-time Algebraic Riccati Equations.
DOI: 10.5220/0005117004040409
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 404-409
ISBN: 978-989-758-039-0

with or without line search, and compares its performance against the state-of-the-art commercial solver `care` from MATLAB Control System Toolbox. The MATLAB solver uses an eigenvalue approach, based on the results in, e.g., (Laub, 1979; Van Dooren, 1981; Arnold and Laub, 1984). One drawback of the Newton's method is its dependence on an initialization, X_0. This X_0 should be stabilizing, i.e., $(\mathrm{op}(A - BK(X_0)), \mathrm{op}(E))$ should be stable, if a stabilizing solution X_s is desired. 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 (Varga, 1981; Hammarling, 1982). However, often such algorithms produce a matrix X_0 and/or the following several matrices X_k, $k = 1, 2, \ldots$ (computed by Newton's 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.

The organization of the paper is as follows. Section 2 starts by summarizing the basic theory and Newton's algorithms for CAREs. A new formula for the ratio of successive residuals is given. 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`. Systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003) are considered. Section 4 summarizes the conclusions.

2 NEWTON'S ALGORITHMS

The following assumptions are made:

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

The conceptual algorithm can be stated as follows:

Algorithm mN: modified 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 $A_k = A - BK_k$.
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 minimizing the squared Frobenius norm of the residual, $\|\mathcal{R}(X_k + tN_k)\|_F^2$, with respect to t.
5. Update $X_{k+1} = X_k + t_k N_k$.

END

Standard Newton's algorithm, briefly referred to as Algorithm sN below, 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, Algorithm mN often outperforms Algorithm sN.

If the assumptions above hold, and X_0 is stabilizing, then the iterates of the Algorithm sN satisfy (Benner and Byers, 1998):

(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

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

If, in addition, $(\mathrm{op}(E)^{-1}\mathrm{op}(A), \mathrm{op}(E)^{-1}B)$ is stabilizable and $t_k \ge t_L > 0$, for all $k \ge 0$, then the iterates of the Algorithm mN 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) Quadratic convergence in a neighbourhood of X_s.

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

Therefore, Algorithm mN does not ensure monotonic convergence of the iterates X_k in terms of definiteness, contrary to Algorithm sN. But Algorithm mN achieves monotonic convergence of the residuals to 0, which is not true for Algorithm sN.

We find now a relation between consecutive residuals $\mathcal{R}(X_k)$ for Algorithm sN. For convenience, we take $E = I_n$, $L = 0$, and $\mathrm{op}(M) = M$. By definition,

$$\begin{aligned} \mathcal{R}(X_k) &= A^T X_k + X_k A - X_k G X_k + Q, \\ -\mathcal{R}(X_k) &= A_k^T N_k + N_k A_k \\ &= A^T N_k + N_k A - X_k G N_k - N_k G X_k, \end{aligned}$$

where we have used the formulas from Steps 2 and 3 of Algorithm mN. Summing up the two formulas above, and using X_{k+1} from Step 5 with $t_k = 1$, we get

$$0 = A^T X_{k+1} + X_{k+1} A - X_k G X_k + Q - X_k G N_k - N_k G X_k . \quad (3)$$

Also, by definition,

$$\mathcal{R}(X_{k+1}) = A^T X_{k+1} + X_{k+1} A - X_{k+1} G X_{k+1} + Q, \quad (4)$$

and subtracting (3) from (4), we have

$$\mathcal{R}(X_{k+1}) = -N_k G N_k . \quad (5)$$

It follows that

$$\|\mathcal{R}(X_k)\|_F / \|\mathcal{R}(X_{k+1})\|_F \approx (\|N_{k-1}\|_F / \|N_k\|_F)^2. \quad (6)$$

If Algorithm sN converges, then $\|N_k\|_F$ is a decresing sequence, so the ratios in (6) should be greater than 1. The ratio itself may be taken as a measure of the iterative process speed. Larger ratios imply faster convergence. Numerical evidence has shown that in many cases, when X_0 has a large norm or is far from X_s, these ratios are close to 4 in the initial iterations of the Algorithm sN for $k > 1$.

Practical algorithms differ in many details from the conceptual algorithms above. For instance, it is not always necessary to use the matrix $K(X_k)$. Indeed, if the matrix R is enough well-conditioned with respect to inversion, the original CARE (1), with $\mathrm{op}(M) = M$, can be re-written in the form

$$\mathcal{R}(X) = \widetilde{Q} + \widetilde{A}^T X E + E^T X \widetilde{A} - E^T X G X E, \quad (7)$$

where

$$\widetilde{A} := A - B R^{-1} L^T, \quad \widetilde{Q} := Q - L R^{-1} L^T. \quad (8)$$

If $m < n/2$ it is more efficient to use a factorization of G. Specifically, let $R = L_c^T L_c$, the Cholesky factorization of R, with L_c upper triangular and nonsingular. Therefore, by definition, $G = DD^T$, with $D = BL_c^{-1}$. Then, $A_k = A - GXE^T = A - DD^T XE^T$. This way, $K(X_k)$ is no longer needed, and L has been removed from the formulas. Moreover, the calculations can be organized so that common expressions can be computed just once, to reduce the computational effort per iteration. A solver has been developed based on the principles mentioned above. It can be used for both continuous- and discrete-time systems.

3 NUMERICAL RESULTS

This section summarizes some results of an extensive performance investigation of the solvers based on Newton's method, and reports deviations from the expected behavior. 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 8.2.0.701 (R2013b). A SLICOT-based MATLAB executable MEX-function has been built using MATLAB-provided optimized LAPACK and BLAS subroutines.

Linear systems from the COMPl$_e$ib collection (Leibfritz and Lipinski, 2003) have been used. This collection contains 124 standard continuous-time examples (with $E = I_n$), with several variations, giving a total of 168 problems. All but 17 problems (for systems of order larger than 2000, with matrices in sparse format) have been tried. The performance index matrices have been chosen as $Q = I_n$ and $R = I_m$. The matrix L was always zero. 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 a MATLAB implementation of the algorithm in (Hammarling, 1982), and when this algorithm 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. The stabilization algorithm was tried on 107 unstable systems, and succeeded for 91 examples. The function `care` failed to solve the Riccati equation for example REA4, since it is unstabilizable. This example has been excluded from our tests.

We tried both standard and modified Newton's methods. The modified solver needed more iterations than the standard solver for 10 examples only. The cumulative number of iterations with the modified and the standard solver for all 150 examples was 1657 and 2253, respectively. The mean number of iterations was about 11 and 15, respectively. Figure 1 plots the ratios between the normalized residuals, $\|\mathcal{R}(X_s)\|_F / \max(1, \|X_s\|_F)$, obtained by Algorithm sN and those obtained by `care`, when Algorithm sN was initialized by `care` and the tolerance was set to ε_M. These ratios indicate the increase in the solution accuracy (in terms of normalized residuals) got by Algorithm sN. Using Algorithm mN instead will not have any advantages with this initialization.

Several situations of unexpected behavior are discussed in the sequel.

Loss of Stabilization. Having a stabilizing initialization for Newton algorithm is not a guarantee that the iterates of the Newton process will also be numerically stabilizing. An example was provided by the COMPl$_e$ib system `HF2D9_M256`, with $n = 256$, $m = 2$. Newton process has been initialized by a

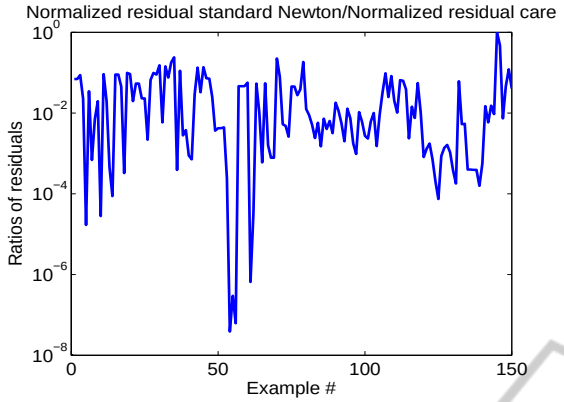

Figure 1: Ratios between the normalized residuals found by standard Newton algorithm and MATLAB `care` for examples from the COMPl$_e$ib collection, using `care` initialization and tolerance ε_M.

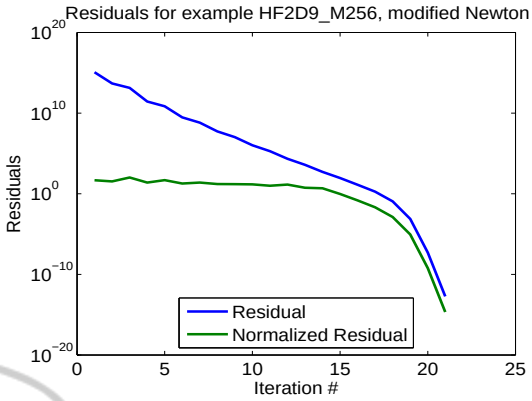

Figure 2: Residual and normalized residual trajectories for COMPl$_e$ib example `HF2D9_M256`, using modified Newton algorithm.

matrix X_0 generated by the algorithm in (Hammarling, 1982). Since $\|X_0\|_F$ is large, slightly larger than $2.475 \cdot 10^{13}$, the residual $\mathcal{R}(X_0)$ is also very large, although $\|A\|_F \approx 198.57$ and $\|B\|_F \approx 5.831$. Indeed, $\|\mathcal{R}(X_0)\|_F \approx 1.166 \cdot 10^{15}$. Consequently, a Lyapunov solution N_0 with big magnitude, $2.39 \cdot 10^{15}$, is obtained at iteration $k = 0$, and the errors propagate to the next iterations. For Algorithm sN, the closed-loop matrix at the second iteration becomes unstable. The iterative process does not converge in 50 iterations, and the instability is inherited. A similar behavior resulted also for the Algorithm mN. Although `HF2D9_M256` was the single example witnessing this difficulty, understanding the reason for the observed bad behavior was considered of great importance. The stabilizing algorithm constructs X_0 as the pseudo-inverse of the solution X_L of a Lyapunov equation. That solution has a low-rank, 16, and the computed X_0 has a large norm. Moreover, the relative residual of the solution N_0, based on X_0, was quite large, about $6 \cdot 10^{-4}$. But it became $2.7 \cdot 10^{-8}$ when the call to the MATLAB function `lyap` was replaced by a call to the SLICOT-based MATLAB function `sllyap` for computing X_L. Then, the modified and standard Newton's algorithms converged in 21 and 27 iterations, respectively. Figure 2 shows the nice residual and normalized residual trajectories obtained using the modified Newton algorithm after changing the initialization. The behavior remained unchanged for the other COMPl$_e$ib examples after the replacement.

Fast Convergence. To understand why the modified Newton algorithm can be faster than the standard Newton algorithm, the behaviour for a simple COMPl$_e$ib example, `AC3` ($n = 5$, $m = 2$), will be used. This is a stable system, hence both Newton algorithms have been initialized with a zero matrix X_0. Figure 3 and Figure 4 show the residual and normalized residual trajectories obtained using modified and standard Newton algorithms, respectively. The use of step sizes equal to 1 produced a significant increase in the two residuals at the first iteration (with values about $1.66 \cdot 10^5$ and $2.77 \cdot 10^2$). The residuals decrease in the following iterations, but a larger number of iterations is needed till the iterative process enters into the region of fast convergence than for the modified Newton algorithm. On the other hand, there is no increase of residuals in Figure 3. This can happen since the first Newton steps are smaller than 1. This is a rather typical behavior.

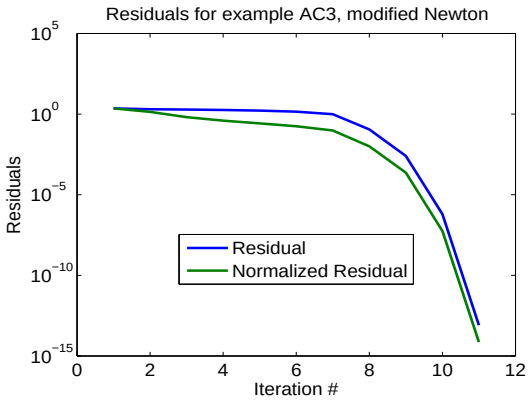

Figure 3: Residual and normalized residual trajectories for COMPl$_e$ib example `AC3`, using modified Newton algorithm.

Stagnation. A difficulty with the modified Newton algorithm is that there might be several consecutive small step sizes, resulting in a slow convergence. To avoid this phenomenon, called *stagnation*, step sizes of value 1 are used instead of the "optimal" ones. This corresponds to a reinitialization of the Newton process. Such a behavior is illustrated for the COMPl$_e$ib example `AC15` ($n = 4$, $m = 2$). Figure 5 shows the

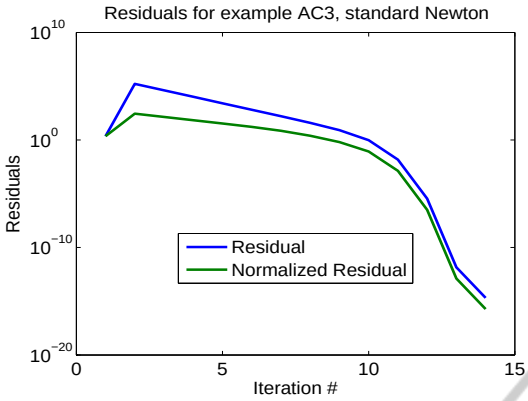

Figure 4: Residual and normalized residual trajectories for COMPl$_e$ib example AC3, using standard Newton algorithm.

residual and normalized residual trajectories obtained using the modified Newton algorithm. The first two step sizes have values about $2.9 \cdot 10^{-4}$ and $1.17 \cdot 10^{-3}$, and the corresponding residuals are very close, 1.7315 and 1.7298. The algorithm then uses a step size 1, which increases the residual to the value $1.94 \cdot 10^4$. The next iteration takes a step size of about 1.997, reducing the residual to 6.18. The previous behavior is repeated once more (with different values), but then the algorithm converges fastly. The standard Newton algorihm has a smoother behaviour from iteration 1 on (similar to that for example AC3 in Figure 4), but it starts with a larger value, $1.19 \cdot 10^7$, and needs one additional iteration.

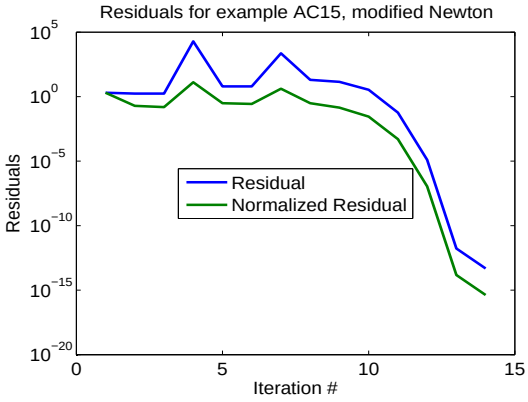

Figure 5: Residual and normalized residual trajectories for COMPl$_e$ib example AC15, using modified Newton algorithm.

Even if, after setting the step to 1, the modified algorithm produces a residual almost as large as for the first iteration of the standard Newton algorithm, usually the next step produces a much lower residual, and so, the number of iterations decreases. Such a behavior has been seen, e.g., for COMPl$_e$ib example JE1 ($n = 30$, $m = 3$), for which, at step 3 of the Newton algorithm, the residual rose to $3.31 \cdot 10^{10}$ (from 5.38 at the previous iteration), but the next residual value was $1.17 \cdot 10^4$. The standard algorithm recorded the maximal residual of $5.43 \cdot 10^{11}$ at the first iteration. The number of iterations for convergence were 11 and 17, respectively.

Slow convergence. The modified Newton algorithm usually converges faster than the standard Newton algoritm. However, there are examples for which the standard algorithm requires less iterations than the modified algorithm. This happened for COMPl$_e$ib examples AC5, AC18, JE2, JE3, DIS5, WEC1, NN11, CM4_IS, CM5_IS, and FS, for which the modified algorithm needed additional 5, 5, 3, 3, 4, 6, 1, 6, 9, and 1 iterations, respectively. To understand the reason of such a behaviour, consider example AC18. Algorithm mN needed 19 iterations, while Algorithm sN needed only 14. The reason is that Algorithm mN uses 7 small, comparable steps, but the residual for all these steps is reduced only by a factor of about 3.5 (from $4.4 \cdot 10^3$ to $1.25 \cdot 10^3$). (On the other hand, using step sizes equal to 1 reduced the residual faster.) A similar behavior appears for COMPl$_e$ib example WEC1, for which Algorithm mN needs 23 iterations, while Algorithm sN needs only 17. The modified algorithm uses 9 small steps (with values in [0.21,0.33]), but the residual is reduced only with a factor of about 3 (from $3.1 \cdot 10^7$ to $1.02 \cdot 10^7$).

This slow convergence is not discovered by the implemented stagnation detection procedure. A possible way to improve such a behaviour is to monitor the succesive reduction factors, defined as ratios between two consecutive residuals. If the current residual is high (e.g., higher than 1000) and the mean value of reduction factors over a certain number of consecutive preceding iterations (say, 4) is lower than a certain value (e.g., 4), then a step size of 1 might improve the convergence rate. This procedure has not yet been tried. The rationale for choosing the values above is as follows. First, such a procedure is not needed when the residuals are small, since then few iterations will be sufficient to achieve convergence. Moreover, for Algorithm mN, the reduction factors may have a large variation from one iterate to the next one, and therefore a moving window of more than two or three iterations is needed to have a useful value for the mean. Finally, the mean value of 4 is a reasonable selection, since it was observed that this is the usual value of the ratios for the initial part of the standard Newton algorithm. Indeed, this was the case for over 90 out of 150 tested COMPl$_e$ib examples. Other 30 examples had ratios larger than 4, while the remaining examples had either lower values or converged too quickly.

Figure 6 and Figure 7 show the ratios between

successive residuals for the modified and standard Newton algorithms for example AC4. The ratios have value 4 for 14 iterations of Algorithm sN. The first ratio, $\|\mathcal{R}(X_1)\|_F/\|\mathcal{R}(X_2)\|_F$ is not included, since usually $\|\mathcal{R}(X_2)\|_F$ is much larger than $\|\mathcal{R}(X_1)\|_F$, so it is not relevant for our purposes. Clearly, the modified Newton algorithm achieves bigger ratios, and so it converges faster than standard algorithm.

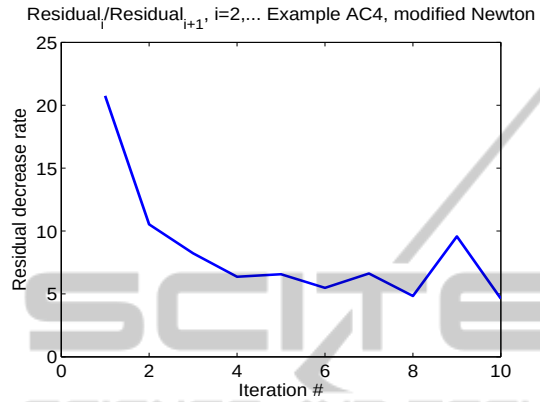

Figure 6: The ratio of successive residuals for COMPl$_e$ib example AC4, using modified Newton algorithm.

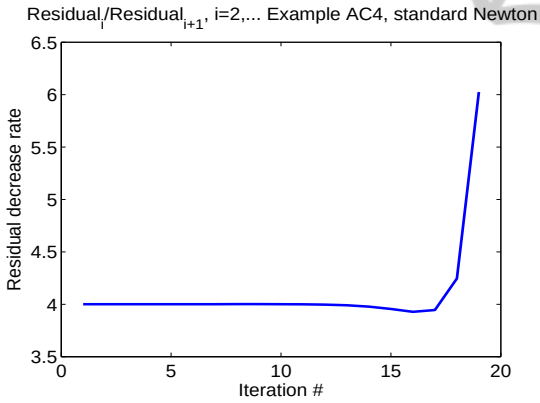

Figure 7: The ratio of successive residuals for COMPl$_e$ib example AC4, using standard Newton algorithm.

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. 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. 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.

REFERENCES

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

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. 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.

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

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

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.

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, Dep. of Mathematics, University of 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* 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.

Sima, V. (1996). *Algorithms for Linear-Quadratic Optimization*. Marcel Dekker, Inc., New York.

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

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