ACCURATE LINEAR QUADRATIC OPTIMIZATION IN AVIATION AND SPACE APPLICATIONS

Vasile SIMA

National Institute for Research & Development in Informatics
Bucharest, Romania (vsima@ici.ro)

DOI: 10.19062/2247-3173.2018.20.37

Abstract: *Linear quadratic Gaussian (LQG) optimization is often used in aviation and space applications. Over three dozens of examples for such applications from the COMPleib benchmark collection are used in this paper to investigate the performance of a new Newton-type algorithm to solve LQG problems. The algorithm efficiency and its accuracy, measured in terms of normalized and relative residuals of computed solutions of algebraic Riccati equations (AREs), are analyzed. Various stabilizing initializations, including that provided by the state-of-the-art MATLAB solver, are considered. The numerical results strongly recommend this algorithm especially for improving approximate solutions computed using other approaches.*

Keywords: *algebraic Riccati equations, linear quadratic optimization, numerical algorithms, optimal control*

1. INTRODUCTION

Algebraic Riccati equations (AREs) are cornerstones for control theory and its practical applications. Many control systems analysis and design procedures require their solution. AREs are the main topic of Linear Quadratic Gaussian (LQG) optimization, involved in optimal control and estimation problems. Such equations appear in various domains, including model reduction, optimal filtering, guidance, (robust) control, etc. Many applications are encountered in the aerospace domain. Actually, control theory recorded a strong development during and after the second World War, mainly to support such applications.

In general, optimization needs powerful computational tools and simulation techniques. Of major importance for numerical calculations are reliability, efficiency, and accuracy of the results. This is due to several reasons, including the limited precision of the calculations, and the need for having the results as quickly as possible (especially, for real-time applications). Moreover, the hidden nature of the intermediate results, from which the returned solution is obtained, requires guarantees on their correctness.

There are several formulas for AREs, depending on the system involved. Of interest in this paper is the continuous-time ARE (CARE) for standard systems, defined by

$$R(X) := A^T X + XA - XBR^{-1}B^T X + Q = 0, \tag{1}$$

where X is the unknown matrix, A and Q are $n \times n$, B is $n \times m$, R is $m \times m$, with Q, R, and X symmetric matrices ($Q = Q^T$, $R = R^T$, $X = X^T$), Q positive-semidefinite ($Q \geq 0$), R positive definite ($R > 0$), hence R nonsingular. The matrices A and B are the state and input (control) matrices of the dynamic system $\mathrm{d}x/\mathrm{d}t = Ax(t) + Bu(t)$, $x(0) = x_0$, while Q and R are the state and input weighting matrices of the performance index, which should be minimized along the system trajectories. With suitable assumptions, such as, the pair (A, B) is stabilizable, and (A, D) detectable, where D is a maximal rank factor of Q, i.e., $D^T D = Q$ and rank(D) = rank(Q), then (1) has a unique positive-semidefinite stabilizing solution, $X^* = X^{*T} \geq 0$, and the optimal control trajectory is given by the state feedback law $u^*(t) = -R^{-1}B^T X^* x(t)$ [1]. This law ensures the stability of controlled system. Specifically, the "closed-loop" system matrix, defined by $A - BR^{-1}B^T X^*$ is stable, that is, all its eigenvalues have negative real parts. Defining $K(X) = R^{-1}B^T X$, then $K(X^*)$ is the controller optimal gain matrix. By a suitable selection of the weighting matrices Q and R, the closed-loop dynamics can be modified to satisfy certain performance criteria, including fast transient response, trajectory following, disturbance rejection, etc.

Dual to the control problem is the estimation problem, which aims to find proper values of some parameters, or of the state of a dynamic system, using observers or filters, like Kalman filter. Many theoretical results have been extended to more general classes of systems, including periodic systems, nonlinear systems, discrete-event systems, etc.

The solutions of a CARE are the matrices $X = X^T$ for which $\mathcal{R}(X) = 0$. When Y is not a solution of (1), then $\mathcal{R}(Y)$ differs from the zero matrix; $\mathcal{R}(Y)$ is called the *residual* of (1) in Y. The Frobenius norm of $\mathcal{R}(Y)$, $||\mathcal{R}(Y)||_F$, is a measure of the error in Y with respect to the solution X.

The literature regarding theory and numerical solution of AREs and their practical applications is vast. Several monographs, e.g., [1-5], address various theoretical and practical issues. There are many techniques and algorithms to compute the optimal solution X^* (see, e.g., [4,5]). Both direct or iterative algorithms have been proposed. The first class contains the (generalized) Schur techniques, e.g., [6-8]. The second class has several categories, including matrix sign function techniques, e.g., [9,10], Newton techniques [6,11], doubling algorithms [12,13], or recursive algorithms [14]. Often used is the direct procedure which computes a basis U of the stable invariant subspace of a Hamiltonian $2n \times 2n$ matrix H built using A, B, Q, and R^{-1}. If R is ill-conditioned with respect to inversion, then the obtained matrix H and, therefore, the computed solution X**,** will be inaccurate. Therefore, in such a case, it is preferable to use an extended matrix pencil of order $2n + m$ instead of H, and compute the associated stable right deflating subspace. This procedure, using either the matrix H, or the extended matrix pencil, is implemented in the state-of-the-art MATLAB function **care**. This matrix pencil can be rewritten as a structured, skew-Hamiltonian/Hamiltonian (sHH) pencil [15], and the optimal problem can be solved by structure-exploiting algorithms [16]. Software implementations for sHH pencils have also been included in the Subroutine Library for Control Theory (SLICOT) [17] (www.slicot.org). Applications in optimal and robust control have been described, e.g., in [18,19].

Newton's method for solving AREs has been considered by many authors, for instance, [3-6]. But matrix sign function method for AREs [9,10] is actually a specialization of Newton's method for computing the square root of the identity matrix of order $2n$.

Newton's method is best used for iterative improvement of a solution, or as a defect correction method [20], delivering the maximal possible accuracy when starting from a good approximate solution. Moreover, it may be preferred in implementing certain fault-tolerant systems, which require controller updating [21].

2. MODIFIED NEWTON ALGORITHM

The algorithmic variants considered in the sequel for CAREs are extensions of Newton's method, which employ a line search procedure attempting to reduce the residual along the Newton direction. The conceptual algorithm can be stated as follows [22]:

Algorithm NCARE: Modified Newton method for CARE

Input: The coefficient matrices A, B, Q, R, and an initial stabilizing matrix $X_0 = X_0^T$.

Output: The approximate solution X_k of CARE (1).

```
FOR $k = 0,1,\dots,k_{max}$, DO
    1. Compute $R(X_k)$. If (non)convergence is detected, return $X_k$ and/or a
       warning or an error value.
    2. Compute $K_k := K(X_k)$ and $A_k$, where $A_k = A - BK_k$.
    3. Solve in $N_k$ the Lyapunov equation $A_k^T N_k + N_k A_k = -R(X_k)$.
    4. Find a step size $t_k$ which minimizes $||R(X_k + tN_k)||_F$ with respect to $t$.
    5. Update $X_{k+1} = X_k + t_k N_k$.
END
```

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

With usual assumptions (e.g., stabilizability of the system pair (A, B), and existence and uniqueness of the stabilizing solution X^*), if X_0 is stabilizing, then the iterates of the Algorithm NCARE with $t_k = 1$ have the following properties [22]:

(a) All matrices X_k are stabilizing.

(b) $X^* \leq \cdots \leq X_{k+1} \leq X_k \leq \cdots \leq X_1$.

(c) $\lim_{k\to\infty} X_k = X^*$.

(d) There is a constant $\gamma > 0$ such that $\|X_{k+1} - X^*\| \leq \gamma \|X_k - X^*\|^2$, $k \geq 1$.

Note that the global quadratic convergence at item (d) does not hold for $k = 0$, involving the iterates X_0 and X_1. The line search variant does not ensure the monotony of the sequence $\{X_k\}$ in terms of definiteness, as in (b), but the convergence of the residual sequence to the zero matrix. The other properties hold; in addition, $\lim_{k\to\infty} t_k = 1$.

More general algorithms for generalized or discrete-time systems, possibly including a state and input cross weighting matrix, are dealt with, e.g., in [22,23], and are implemented in a new Newton-type solver.

The basic stopping criterion for the iterative process of the Newton solver is expressed in terms of a *normalized residual*, $r_k := r(X_k) := \|R(X_k)\|_F / \max(1, ||X_k||_F)$, and a tolerance τ. If $r_k \leq \tau$, 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.

For systems with very large norms of the matrices A, B, and/or Q, and a small norm of the solution X^*, the stopping criterion involving r_k might not be satisfied in a reasonable number of iterations (or never, due to accumulated rounding errors), while an acceptable approximate solution might be much earlier available. Therefore, the MATLAB-style *relative residual*, $r_r(X_k)$, which is the ratio of $\|R(X_k)\|_F$ and the sum of Frobenius norms of the matrix terms of (1), 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 r_k. 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.

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. Specifically, t_k is set to 1 when stagnation, or other criteria of slow convergence, are detected. This is equivalent with a restart of the standard Newton algorithm, which is theoretically guaranteed to converge from any stabilizing initialization. On the other hand, after such a reset, the residual norm might increase, sometimes significantly, but fewer unit steps are generally needed than for a stagnating line search procedure. Anyhow, the residual increase is smaller than what might appear in the beginning of the iterative process if only standard steps would be used. Consequently, this strategy is very attractive.

Other line search strategies, including *combined* or *hybrid strategies* have also been investigated. Specifically, in the combined strategy, line search is employed in the beginning of the iterative process, but the algorithm switches to the standard method when the normalized residual is smaller than a specified (or default) tolerance. In the hybrid strategy, both standard Newton step and the step corresponding to the line search procedure are computed, and the step which gives the smallest residual is selected at each iteration.

3. NUMERICAL RESULTS

This section presents some results of a performance investigation of the new Newton solver, developed by the author. 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 $\varepsilon_M \approx 2.22\times10^{-16}$, using Windows 7 Professional (Service Pack 1) operating system (64 bit), Intel Visual Fortran Composer XE 2015 and MATLAB 8.6.0.267246 (R2015b). A MATLAB executable MEX-function has been built using MATLAB-provided optimized LAPACK [24] and BLAS subroutines.

The results reported here have been obtained for linear systems modelling aerospace applications from the COMPl$_e$ib collection [25], which contains 124 standard continuous-time models. Specifically, the examples tried are listed below, where the notation *n* = [a..b] means that *n* has a minimum value a and a maximum value b, and *p* is the number of system outputs:

- Aircraft models (AC1−AC18), with n = [4..55], m = [1..4], p = [2..4];
- Helicopter models (HE1 − HE7), with n = [4..20], m = [2..4], p = [1..6];
- Jet engine models (JE1 − JE3), with n = [21..30], m = 3, p = [3..6];
- Academic models (NN5, NN15, NN16), with n = [3..8], m = [1..4], p = [2..4];
- Flexible satellite model (FS), with n = 5, m = 1, p = 3;

- Space structure models (DLR1 − DLR3), with n = [10..40], $m = p = 2$;
- International Space Station component (ISS1, ISS2), with n = 270, $m = p =$ 3;
- Reduced order models (ROC2, ROC5), with n = [7..10], m = [2..3], p = [3..5].

A brief description is given below. More details are given in [25] and the references therein. AC1 and AC2 model the linearized vertical-plane dynamics of an aircraft. AC3 models an L-1011 aircraft in cruise flight conditions, while AC4 describes an autopilot control problem for an air-to-air missile. AC5 describes the motion of a Boeing B-747 aircraft flying at 20000 ft with a speed of Mach 0.8. AC6 is an L-1011 aircraft model; AC7 and AC8 model the motion of a transport aircraft at 35000 ft, with Mach 0.57 and with the center of gravity at the most aft location, and at the aft location, respectively. AC9 is a variation of AC8 with an additional state and four inputs instead of one. AC10 is an aeroelastic model of high order describing a modified Boeing B-767 airplane at flutter condition. AC11 is a linearized model of an CVC-type aircraft. AC12 − AC14 define the linearized equations of motion for the longitudinal dynamics of an ASTOVL (Advanced Short Take-Off and Vertical Landing) aircraft, with increasing orders. AC15 and AC16 model a supersonic transport aircraft flying at Mach 2.7. AC17 is a model of the lateral axis dynamic of a L-1011 aircraft, and AC18 is a reduced order model of AC10.

HE1 describes the longitudinal motion of a VTOL helicopter at flying speed of 135 knots, while HE2 models the longitudinal-vertical motion of an AH-64 helicopter at 130 knots. HE3 represents the linearized dynamics of a Bell 201A-1 helicopter, and HE4 − HE7 are variations of a model for a twin-engine, multi-purpose military helicopter.

JE1 represents a J-100 jet engine, and JE2 and JE3 are variations of a model for a Rolls-Royce 2-spool reheated turbofan for a military aircraft.

NN5 is a model of a Saturn V booster, while NN15 is a space backpack model, and NN16 describes a large space structure.

FS presents the dynamics of a flexible satellite, deduced from a second order model (with damping and stiffness matrices). DLR1−DLR3 are variations of a model describing the active vibration damping of large flexible space structures. ISS1 and ISS2 are models of a component of the International Space Station.

Finally, ROC2 models the same aircraft as AC7, but for an altitude of 25500 ft at Mach 0.87, and ROC5 describes a free gyro-stabilized mirror system used to stabilize the sensors mounted on vehicles subjected to vibrations, like aircrafts and helicopters.

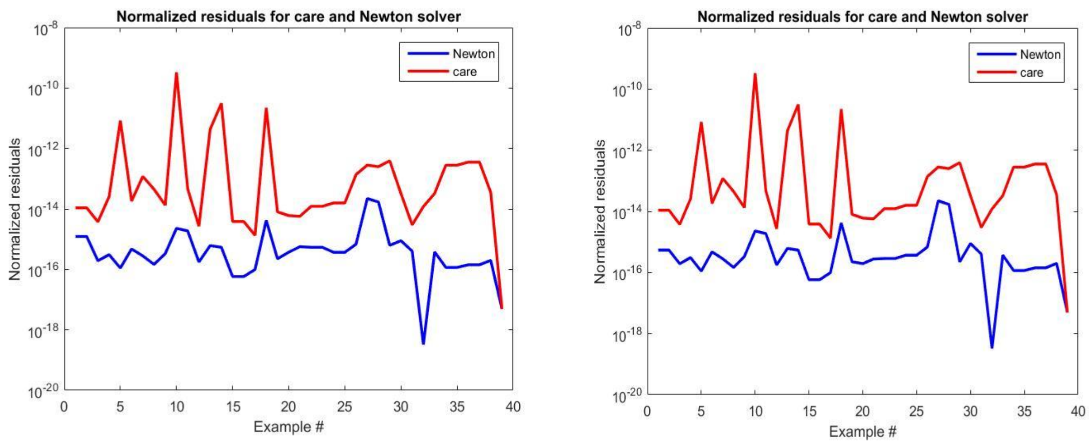
The algebraic Riccati equations have been solved for all these 39 examples, using weighting matrices set to identity, $Q = I_n$, $R = I_m$. The purpose of our study was not to find suitable weighting matrices for solving specific optimal control problems, but to investigate the performance of the new solver.

In one set of tests, Newton solver was initialized by the solution computed by the state-of-the-art MATLAB function **care**. The tolerance τ has been set either to the default value, or to ε_M. With the default value, Newton solver needed just one iteration to achieve the required accuracy for all examples, except ROC5 (numbered 39), for which it returned before finishing the first iteration, because $\|R(X_0)\|_F$ was already below the tolerance value. With tolerance ε_M, Newton solver needed 0 iterations for ROC5, 2 iterations for AC18, HE2 − HE5, and DLR1, 3 iterations for NN5, 4 iterations for AC1 and AC2, and still 1 iteration for the remaining 29 examples.

Figure 1 displays the normalized residuals of CARE solutions for the mentioned 39 examples from the COMPl$_e$ib collection, computed using MATLAB function **care** and standard Newton solver, with **care** initialization and either default tolerance, in part (a), or tolerance ε_M, in part (b). Slightly more accurate results than in part (a) are obtained in part (b) for examples mentioned above, numbered 1, 2, 18, 20:23, 29, and 33, which needed more than one iteration. For all examples, but the last one (ROC5, numbered 39), Newton solver was more accurate than **care**, and it improved the normalized residuals sometimes with several orders of magnitude. (Note that the ordinate axes are in a logarithmic scale.)

In the same way as in Fig. 1, Fig. 2 plots the MATLAB-style relative residuals. Using a tolerance set to ε_M, slightly more accurate results are obtained for the same examples as above. It is worth mentioning that Newton solver obtained relative residuals close to the limiting accuracy ε_M of the computer, or even smaller than ε_M, while **care** sometimes returned much larger residuals. Moreover, the variation of these values is in a much larger interval for **care** than for Newton solver, which shows a more uniform behavior. For most examples, this improvement is obtained in just one Newton iteration.

Similarly, Fig. 3 shows the corresponding elapsed CPU times for the two solvers. Part (a) of Fig. 3 compares **care** and standard Newton solver, while part (b) also includes modified Newton solver; moreover, balancing the matrices A_k was either used or not before solving Lyapunov equations for both variants. (Balancing may reduce the 1-norm of a matrix and improve accuracy of the computed results.) Clearly, balancing, but especially line search (LS), increases somewhat the computing time. Since the normalized and relative residuals for all these four options were the same with **care** initialization, it is recommended to use standard Newton variant (STD) in such a case. Since very few iterations are most often needed, the CPU time for Newton solver is a small fraction of that for **care**.



(a) (b)

FIG. 1. Normalized residuals for 39 COMPl$_e$ib examples using MATLAB function **care** and standard Newton solver with: (a) default tolerance; (b) tolerance;slightly more accurate results than in part (a) are obtained in part (b) for examples numbered 1, 2, 18, 20:23, 29, and 33

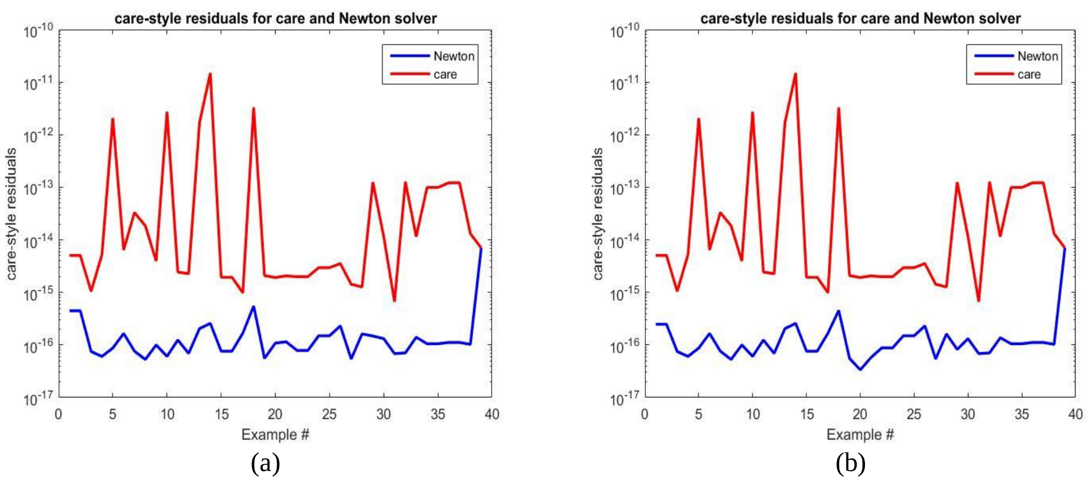


FIG. 2. Relative, **care**-style residuals for 39 COMPl$_e$ib examples using MATLAB function **care** and standard Newton solver with: (a) default tolerance; (b) tolerance; slightly more accurate results than in part (a) are obtained in part (b) for examples numbered 1, 2, 18, 20:23, 29, and 33

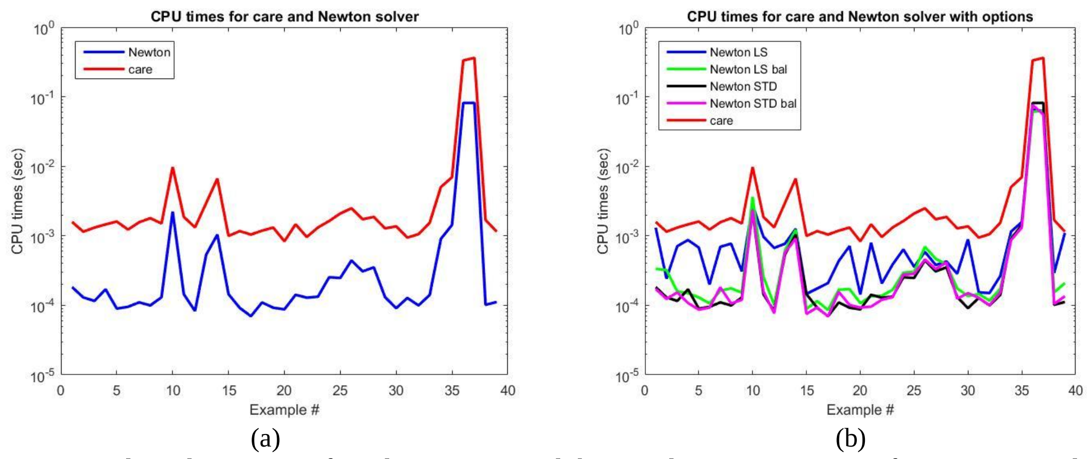


FIG. 3. Elapsed CPU times for solving 39 COMPl$_e$ib examples using MATLAB function **care** and Newton solver with default tolerance; (a) CPU times for **care** and standard Newton solver; (b) CPU times for **care** and Newton solver with various options: line search (LS); LS with balancing (LS bal); standard (STD); STD with balancing (STD bal); option STD is usually the fastest

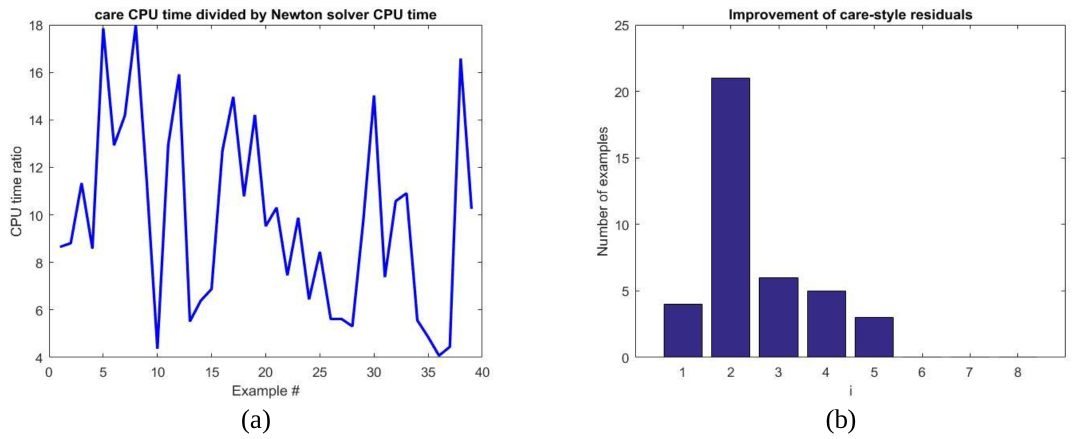
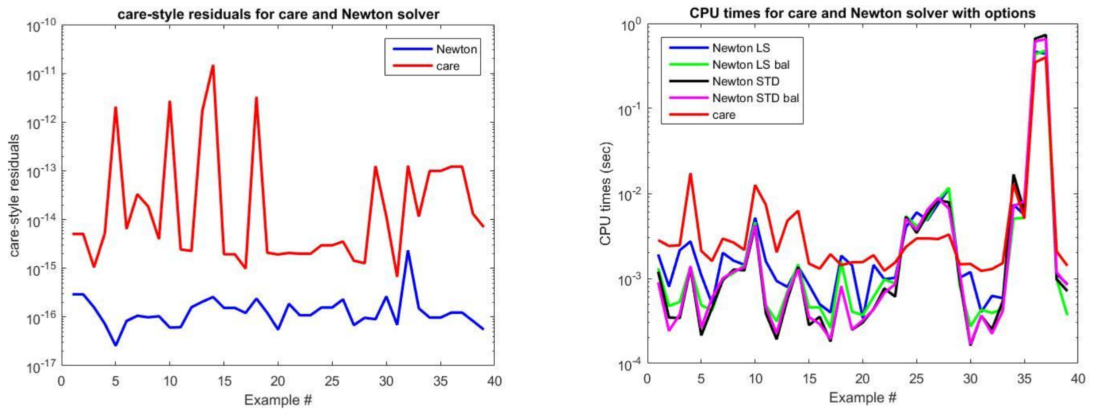


FIG. 4. (a) Ratios of the elapsed CPU times for MATLAB function **care** and standard Newton solver with default tolerance; (b) improvement of **care**-style residuals obtained by standard Newton solver against **care** solver; the height of the i-th bar indicates the number of examples for which the improvement was between i-1 and i orders of magnitude

Part (a) of Fig. 4 plots the ratios of the elapsed CPU time needed by MATLAB function **care** and standard Newton solver. The bar graph from part (b) shows the improvement obtained using standard Newton solver, default tolerance and **care** initialization. Specifically, the height of the i-th vertical bar indicates the number of examples for which the improvement was between i−1 and i orders of magnitude, in comparison to **care**. The number of examples in the five bins are 4, 21, 6, 5, and 3, corresponding to improvements till one order of magnitude for four examples, between one and two orders of magnitude for 21 examples, and so on, and finally between four and five orders of magnitude for 3 examples.

In another set of tests, X_0 was set to a zero matrix, if A was found to be stable; otherwise, an initialization of Newton solver with a matrix computed using the stabilization algorithm in [11] was tried, and when this algorithm failed to deliver a stabilizing X_0 matrix, the solution provided by **care** was used. There are 12 stable examples (AC3, AC6, AC15 −AC17, HE2, JE1, DLR1−DLR3, ISS1, and ISS2), but the other 27 examples are unstable. For 11 examples and default tolerance, one or another variant of Newton's method obtained larger normalized and **care**-style residuals than MATLAB function **care**. These examples are AC3, AC6, JE1 − JE3, DLR1 − DLR3, ISS1, ISS2, and ROC5, and eight of them are stable. But either standard Newton variant (for AC3, AC6, and JE1), or Newton variant with line search (for other examples, but ROC5), was more accurate by one or more orders of magnitude.

Figure 5 (a) plots the MATLAB-style relative residuals for **care** and standard Newton solver with tolerance set to ε_M. Figure 5 (b) shows the elapsed CPU times for **care** and both standard and modified Newton solver with tolerance ε_M, with or without balancing. Standard and modified Newton solvers are more accurate for all examples, but they can be more time consuming than **care** for some examples, which require more iterations with X_0 set to 0, or to the matrix computed by the algorithm in [11]. This happened for examples HE6, HE7, JE1 − JE3, DLR3, ISS1, and ISS2.



(a) (b)

FIG. 5. Relative residuals and elapsed CPU times for solving 39 COMPl$_e$ib examples using MATLAB function **care** and Newton solver with tolerance ε_M and various initializations (either **0**, or provided by the algorithm in [11], or by **care**); (a) relative residuals for **care** and standard Newton solver; (b) CPU times for **care** and Newton solver with various options: line search (LS); LS with balancing (LS bal); standard (STD); STD with balancing (STD bal); for few examples, **care** is the fastest solver

The maximum number of iterations was 29 for JE2 and JE3 with line search variant and for ROC5 with standard variant. Specifically, for ROC5, with unit step sizes, the solver generated an iterate X_1 with a residual of Frobenius norm larger than 10^{14} and needed 26 iterations to reduce it to a value less than 1, and finally to about 10^{-11} (with a corresponding normalized residual of order 10^{-15}). On the other hand, line search variant started with a step size of order 10^{-8} and returned after five iterations with a step size very close to 1, and a residual value of order 10^{-8} (with a corresponding normalized residual of order 10^{-12}). This shows the potential of line search to speed up the convergence rate. A similar behavior was encountered for several other examples.

The mean number of iterations was almost 11 for the line search variant and 14 for the standard one. The mean values of the normalized residuals were about $3.56 \cdot 10^{-13}$ for the line search variant, $9.16 \cdot 10^{-13}$ for the standard variant and $1.03 \cdot 10^{-11}$ for **care.** Examples AC18, JE1, NN5, and ROC5, for line search, and JE2, JE3, DLR1− DLR3, ISS1, and ISS2, for standard variant, contributed significantly to the increase of these means.

The mean CPU time for the line search variant was comparable to (but slightly larger than) that for **care**, while for the standard variant, it was about 152% larger. Balancing option somewhat reduced the differences, and line search with balancing was slightly faster than **care**.

CONCLUSIONS

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

AKNOWLEDGMENT

This work was partially supported by the Institutional research program PN 1819 "Advanced IT resources to support digital transformation processes in the economy and society — RESINFO-TD" (2018), project PN 1819-01-01, "New research in complex systems modelling and optimization with applications in industry, business and cloud computing", funded by the Ministry of Research and Innovation, Romania.

REFERENCES

[1] B. D. O. Anderson and J. B. Moore, *Linear optimal control*, Prentice-Hall, Englewood Cliffs, New Jersey, 1971;
[2] D. A. Bini, B. Iannazzo and B. Meini, *Numerical solution of algebraic Riccati equations*, SIAM, Philadelphia, 2012;
[3] P. Lancaster and L. Rodman, *The algebraic Riccati equation*, Oxford University Press, Oxford, 1995;
[4] V. Mehrmann, *The autonomous linear quadratic control problem. Theory and numerical solution*, Springer-Verlag, Berlin, 1991;
[5] V. Sima, *Algorithms for linear-quadratic optimization*, Marcel Dekker, Inc., New York, 1996;

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

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

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

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

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

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

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

[13]P.-C. Guo, A modified large-scale structure-preserving doubling algorithm for a large-scale Riccati equation from transport theory, *Numerical Algorithms*, vol. 71, no. 3, pp. 541–552, 2016;

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

[15]P. Benner, R. Byers, V. Mehrmann and H. Xu, Numerical computation of deflating subspaces of skew Hamiltonian/Hamiltonian pencils, *SIAM J. Matrix Anal. Appl.*, vol. 24, pp. 165–190, 2002;

[16]P. Benner, V. Sima and M. Voigt, Algorithm 961: Fortran 77 subroutines for the solution of skew-Hamiltonian/Hamiltonian eigenproblems, *ACM Trans. on Math. Softw.*, vol. 42, article 24, pp. 1-26, 2016;

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

[18]P. Benner, V. Sima and M. Voigt, L_∞-norm computation for continuous-time descriptor systems using structured matrix pencils, *IEEE Trans. Automat. Contr.*, vol. AC-57, no. 1, pp. 233–238, 2012;

[19]V. Sima and P. Benner, *Solving SLICOT benchmarks for continuous-time algebraic Riccati equations by Hamiltonian solvers*, in S. Caraman, M. Barbu and R. Solea (Eds), *Proceedings of the 2015 19th International Conference on System Theory, Control and Computing*, pp. 1–6, Cheile Gradistei, Romania, October 14-16, 2015;

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

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

[22]P. Benner, *Contributions to the numerical solution of algebraic Riccati equations and related eigenvalue problems*, Thesys, Fakultät für Mathematik, Technische Universität Chemnitz–Zwickau, Germany, 1997;

[23]V. Sima, *Computational experience with a modified Newton solver for continuous-time algebraic Riccati equations*, in J.-L. Ferrier, O. Gusikhin, K. Madani and J. Sasiadek (Eds.), *Informatics in Control Automation and Robotics*, Lecture Notes in Electrical Engineering, vol. 325, ch. 3, pp. 55-71, Springer International Publishing, Switzerland, 2015;

[24]E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK users' guide: Third edition*, SIAM, Philadelphia, PA, 1999;

[25]F. Leibfritz and W. Lipinski, *Description of the benchmark examples in COMPl$_e$ib 1.0,* University of Trier, Germany, 2003.