Step Size Control for Newton's Method Applied to Algebraic Riccati Equations

Peter Benner * Ralph Byers †

February 28, 1994

Abstract

This paper studies Newton's method and Newton-like methods for solving the algebraic Riccati equation combined with exact line search step size control. The cost of computing the exact line search is small relative to the cost of a Newton step.

1 Introduction

The algebraic Riccati equation (ARE)

$$0 = R(X) = F + A^T X + XA - XGX \tag{1}$$

arises frequently in control problems. Here A, G, and F are known $n \times n$ matrices, F and G are symmetric positive semidefinite, and X is an unknown $n \times n$ matrix. Usually, the desired solution is *stabilizing* in the sense that the eigenvalues of $A - GX$ have negative real parts. Under mild assumptions, such a positive semi-definite, stabilizing solution exists and is unique [12].

The Riccati equation (1) is a nonlinear system of equations. One of the oldest, best studied, numerical methods for solving (1) is Newton's method [6, 9, 12, 13].

Algorithm 1

1. $X_0 :=$ some symmetric initial starting "guess"
2. FOR $k = 0, 1, 2, \ldots$
   - 2.1 $R_k := -R(X_k) = -(F + A^T X_k + X_k A - X_k G X_k)$
   - 2.2 Solve for S_k in the Lyapunov equation $(A - GX_k)^T S_k + S_k(A - GX_k) = R_k$
   - 2.3 $X_{k+1} := X_k + S_k$

Algorithm 1 modifies none of the coefficient matrices A, F, and G and corrects iterate X_k by adding a (hopefully) small step S_k. We prefer Algorithm 1 to some other mathematically equivalent versions of Newton's method, because of its robustness in the presence of rounding errors.

It is well known [6, 9, 12, 13] that if X_0 is chosen such that $A - GX_0$ is *stable* in the sense that the eigenvalues have negative real parts, then the iteration converges to the desired

*Technische Universität Chemnitz–Zwickau, Fachbereich Mathematik, Reichenhainer Str. 41, 09009 Chemnitz, FRG. E-mail: benner@mathematik.tu-chemnitz.de Some of this work was complete while this author was visiting the University of Kansas.

†405 Snow Hall, Department of Mathematics, University of Kansas, Lawrence, KS 66045. E-mail: byers@math.ukans.edu This author was partially supported by National Science Foundation grants INT-8922444, and DMS-9205538, and the Kansas Institute for Theoretical and Computational Science.

stabilizing solution X. Ultimately the convergence is quadratic. At each step $A - GX_k$ is stable. *After* the first step, convergence is monotone.

Although it ultimately converges rapidly, initially, the iteration may converge slowly. Automatic stabilizing procedures like [1] or [14] may give choices of X_0 that lie far from the solution X. Sometimes the first Newton step S_0 is disastrously large and many iterations are needed to find the region of rapid convergence. Computing S_k requires an expensive factorization of $A - GX_k$, so it is attractive to replace S_k with an inexpensive approximation. If the Lyapunov equation is ill-conditioned it may be difficult to compute an accurate Newton step. (This sometimes signals that the Riccati equation is ill-conditioned [3].) If the Newton step can not be calculated accurately, then the usual convergence theory breaks down. Sometimes rounding errors or a poor choice of X_0 cause Newton's method to converge to a non-stabilizing solution. For these reasons, Newton's method is not often used by itself to solve Riccati equations. However, when Newton's method is combined with another, more robust method, it is often able to squeeze out the maximum possible accuracy [4, 8, 7].

Our goal is to restore some robustness to Newton's method and to approximate Newton's methods and to accelerate convergence through step size control. We show that the cost of exact line searches is small compared to the cost of a Newton Step.

2 Step Size Control by Exact Line Search

Line searches are a well understood technique commonly used in numerical methods for optimization [5]. The approach is to replace Step 2.3 in Algorithm 1 by $X_{k+1} = X_k + t_k S_k$ where t_k is a real scalar chosen to minimize some measure of the error. One idea is to minimize the Frobenius norm of the next residual $R(X_{k+1}) = R(X_k + t_k S_k)$. The line search is said to be *exact* if t_k is an exact (as opposed to approximate) minimizer.

Let $T_k = S_k G S_k$ and $V_k = (A - GX_k)^T S_k + S_k(A - GX_k)$. The Frobenius norm of a matrix Z is defined by $\|Z\|_F^2 = \mathrm{trace}(Z^T Z)$. Using the fact that for any symmetric matrix Z, $\|Z\|_F^2 = \mathrm{trace}(Z^2)$, it is easy to derive that finding t_k to minimize $\|R_{k+1}\|_F$ is equivalent to minimizing

$$
\begin{aligned}
P(t) \quad = \quad & \mathrm{trace}\left(R(X_k)^2\right) \\
& + 2t \cdot \mathrm{trace}\left(R(X_k)V_k\right) \\
& + t^2 \cdot \left(\mathrm{trace}\left(V_k^2\right) - 2\mathrm{trace}\left(R(X_k)T_k\right)\right) \\
& - 2t^3 \cdot \mathrm{trace}\left(T_k V_k\right) \\
& + t^4 \cdot \mathrm{trace}\left(T_k^2\right).
\end{aligned} \tag{2}
$$

This is a polynomial of degree four. Since $\mathrm{trace}\left(T_k^2\right) > 0$ it has either one minimum and no maxima or two minima and one maximum. In the next section we give some proposals how to choose between the two minima and also consider the problem of whether the modified algorithm still converges to the desired positive semidefinite solution.

But first we want to discuss if it is worth doing the computations for finding t_k. Consider the following modification of Algorithm 1.

Algorithm 2

1. $X_0 :=$ some symmetric initial starting "guess"

2. FOR $k = 0, 1, 2, \ldots$

   2.1 $R_k := -R(X_k) = -(F + A^T X_k + X_k A - X_k G X_k)$

   2.2 Solve for S_k in the Lyapunov equation $(A - GX_k)^T S_k + S_k(A - GX_k) = R_k$

   2.3a $T_k := S_k G S_k$; $V_k := (A - GX_k)^T S_k + S_k(A - GX_k)$.

   2.3b Select a minimizer t_k of $P(t)$ in (2).

   2.3c $X_{k+1} := X_k + t_k S_k$

Each iteration in Algorithm 1 solves a Lyapunov equation and does several matrix multiplies. On conventional serial computers, the Lyapunov equation solution costs 10–12 times the work of a matrix multiply [2], so Algorithm 1 does work approximately equivalent to (say) 15–17 matrix multiplies. In the modified algorithm, we must, in addition, compute T_k and V_k at the cost of another three matrix multiplies. (The coefficient $A - GX_k$ is needed for the Lyapunov equation and need not be recomputed. Because of this and the symmetry of S_k, V_k costs only one extra matrix multiply. Symmetry can be used to trim some of the work computing T_k [2], so T_k and V_k actually may cost less than three matrix multiplies.) Only the diagonal entries of the matrix products in (2) are needed, so once T_k, V_k are computed, the coefficients of $P(t)$ cost only a negligible $O(n^2)$ flops. Algorithm 2 does about 15%–20% more work per iteration than the conventional Newton's method. (We found that our MATLAB implementation of Algorithm 2 did only 9%–10% more work than the implementation of Algorithm 1. Neither implementation took full advantage of symmetry.)

Note also that the cost of T_k and V_k is approximately the same as the cost of calculating the residual $R(X_k) = F + A^T X_k + X_k A - X_k G X_k$, so the cost of the exact line search is competitive with other strategies that require one or more extra residual calculation.

We do need some additional work space to store T_k and V_k, i.e. $n(n+1)/2$ storage for each of the matrices T_k and V_k. However, T_k and V_k are computed implicitly by Newton's method.

The following version of Newton's method accomplishes the line search at no significant extra work over Algorithm 1. Unfortunately, its rounding error properties are not as good as Algorithms 1 and 2.

Algorithm 3

1. $S_0 := X_0; \quad X_0 := 0; \quad A_0 := A - GS_0; \quad F_0 := R(S_0)$

2. FOR $k := 0, 1, 2, \ldots$

   2.1 Solve for S_k in the Lyapunov equation $A_k^T S_k + S_k A_k := -F_k$

   2.2 $T_k := S_k G S_k; \quad V_k := A_k^T S_k + S_k A_k$

   2.3 Select a minimizer t_k of $P(t)$ in (2).

   2.4 $X_{k+1} := X_k + t_k S_k$

   2.5 $A_{k+1} := A_k - t_k G_k S_k; \quad F_{k+1} := F_k + t_k V_k - t_k^2 T_k$

3 The Choice of t_k

Algorithms 2 and 3 recast the nonlinear equation (1) as a non-linear least squares problem. The convergence theory for this approach is well known and largely satisfactory. (For example, see [5, § 6.5].)

However, convergence — even convergence to a solution is not sufficient. Usually the symmetric, positive semi-definite, stabilizing solution is required. We would like Algorithms 2 and 3 to have the same kind of guaranteed convergence to the desired solution as Algorithm 1 has when $A - GX_0$ is stable.

In order to have convergence behavior that is similar to the ordinary Newton iteration [9, 12, 13], we need $X_{k+1} \geq 0$, and $A - GX_{k+1}$ stable for all k. At this writing, our understanding of this question is incomplete, but here is a partial result. We will use the notation $M \geq 0$ to indicate that the matrix M is symmetric and positive semi-definite.

THEOREM 3.1. *If $A - GX_0$ is stable, the initial residual $R(X_0) \leq 0$, and $0 < t_k \leq 1$ for all k, then $A - GX_k$ is stable, $R(X_k) \leq 0$ and $0 \leq X_{k+1} \leq X_k$ for all k. If in addition, t_k is bounded away from zero, then Algorithms 2 and 3 converge to the desired positive semidefinite solution of (1).*

Proof. The proof is an extension of arguments in [9, 12, 13].

The above theorem is disappointing. We are certainly interested in $t_k > 1$ as well as $t_k < 1$. Numerical computations show that the t_k given by the exact line search is contained in (0, 1] for the early steps but later during the iteration we mostly get $1 < t_k < 2$.

4 Numerical Examples

We compared Algorithm 1 with Algorithm 3 on several randomly generated examples and some of the examples from [10]. The stabilizing initial guess X_0 was generated by the method described in [1, 6]. We terminated the iteration when $\|R(X_k)\|_F$ dropped below $\epsilon n^2 \|H\|$ where ϵ is the unit round and $H = \begin{bmatrix} A^T & G \\ F & -A \end{bmatrix}$. All computations were done under Matlab version 4.0a [11] on SGI Iris Indigo R4000 at the Institut für Geometrie und Praktische Mathematik of the Technical University of Aachen, Germany.

Although we could prove convergence to the stabilizing solution for the step size control iteration only for $t_k \in (0, 1]$ we allowed $0 < t_k < 2$. If we couldn't find a minimizer t_k of $R(X_k + t_k S_k)$ in the interval (0, 2), then we arbitrarily set $t_k = 1$. This case occurred only once or twice during late stages of the iteration. In no case did Algorithm 3 converge to a solution other than the required symmetric positive definite one.

Example 1 We generated some 20 random examples of algebraic Riccati equations with order up to $n = 100$ with $G = BB^T$, $F = C^T C$, and the elements of A, B and C uniformly distributed in the interval [0, 1]. The Algorithm 3 never used more iterations than Algorithm 1. The number of iterations saved increased with increasing order of the ARE. For an example of order $n = 50$, Algorithm 3 used 10 iterations as opposed to 15 iterations for Algorithm 1. For an example of order $n = 100$, Algorithm 3 used 11 iterations as opposed to 17 for Algorithm 1.

Example 2 Here we used

$$A = \begin{bmatrix} -2 & 1 & 0 & & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & & & & & 1 \\ 1 & 0 & \dots & & 0 & 1 & -2 \end{bmatrix}, \quad G = F = I_n.$$

This example is considered in [10] for $n = 64$. The solution of the ARE is a circulant matrix and the closed loop spectrum contains two simple eigenvalues and 31 of multiplicity

2. Algorithm 3 used 6 iterations whereas Algorithm 1 used 7.

References

[1] E. S. Armstrong, *An extension of Bass' algorithm for stabilizing linear continuous constant systems*, IEEE Trans. Automat. Control, AC–20 (1975), pp. 153–154.

[2] R. Bartels and G. Stewart, *Solution of the matrix equation $AX + XB = C$: Algorithm 432*, Comm. ACM, 15 (1972), pp. 820–826.

[3] R. Byers, *Numerical condition of the algebraic Riccati equation*, Contemp. Math., 47 (1985), pp. 35–49.

[4] ——, *Solving the algebraic Riccati equation with the matrix sign function*, Linear Algebra Appl., 85 (1987), pp. 267–279.

[5] J. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[6] S. J. Hammarling, *Newton's method for solving the algebraic Riccati equation*, NPL Report DITC 12/82, National Physical Laboratory, Teddington, Middlesex TW11 OLW, U.K., 1982.

[7] C. Kenney, A. Laub, and M. Wette, *A stability-enhancing scaling procedure for Schur-Riccati solvers*, Sys. Control Lett., 12 (1989), pp. 241–250.

[8] ——, *Error bounds for Newton refinement of solutions to algebraic Riccati equations*, Math. Control, Signals, Sys., 3 (1990), pp. 211–224.

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

[10] A. Laub, *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Control, AC-24 (1979), pp. 913–921. (see also *Proc. 1978 CDC (Jan. 1979)*, pp. 60-65).

[11] *MATLAB*. The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Mass, 01760, 1984–94.

[12] V. Mehrmann, *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution*, no. 163 in Lecture Notes in Control and Information Sciences, Springer-Verlag, Heidelberg, July 1991.

[13] N. Sandell, *On Newton's method for Riccati equation solution*, IEEE Trans. Automat. Control, AC-19 (1974), pp. 254–255.

[14] V. Sima, *An efficient Schur method to solve the stabilization problem*, IEEE Trans. Automat. Control, AC-26 (1981), pp. 724–725.