Computational Complexities and Storage Requirements of Some Riccati Equation Solvers

A. V. Ramesh* and Senol Utku†
Duke University, Durham, North Carolina
and
John A. Garba‡
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

The linear optimal control problem of an *n*th-order time-invariant dynamic system with a quadratic performance functional is usually solved by the Hamilton-Jacobi approach. This leads to the solution of the differential matrix Riccati equation with a terminal condition. The bulk of the computation for the optimal control problem is related to the solution of this equation. There are various algorithms in the literature for solving the matrix Riccati equation. However, computational complexities and storage requirements as a function of numbers of state variables, control variables, and sensors are not available for all these algorithms. In this work, the computational complexities and storage requirements for some of these algorithms are given. These expressions show the immensity of the computational requirements of the algorithms in solving the Riccati equation for large-order systems such as the control of highly flexible space structures. The expressions are also needed to compute the speedup and efficiency of any implementation of these algorithms on concurrent machines.

Introduction

THE linear optimal control problem of an *n*th-order time-invariant dynamic system with a quadratic performance functional is usually solved by the Hamilton-Jacobi approach. This involves the solution of the differential matrix Riccati equation with a terminal condition. The bulk of the computation for the optimal control problem is related to the solution of this equation.

The solution of the matrix Riccati equation also arises in the filtering problem. Various algorithms have been proposed to solve this equation.[1-38] There has also been a lot of work concerning the numerical and computational aspects of these algorithms.[1-38]

In this work, some of these algorithms are analyzed for their computational complexity and storage requirements. The computational complexity and storage requirements are given as functions of the parameters of the system. Such estimates give a good account of the immensity of the computational requirements of these algorithms in solving the Riccati equation for large-order systems, such as those that are encountered in the vibration control of large space structures. These estimates are also needed to compute the speedup and efficiency of any implementation of these algorithms on concurrent machines. Also, computation time plays an important role in any real-time control application for large-order systems.

The algorithms are grouped under two main classes: algorithms for the solution of the differential matrix Riccati equation and algorithms for the solution of the algebraic matrix Riccati equation. Since most of these methods require iterations to find the solution, complexity estimates per iteration are given. The estimates are useful in showing the superiority of one algorithm over another in speed and storage requirements.

Received April 3, 1987; revision received Sept. 10, 1987.

*Graduate Student, Department of Civil and Environmental Engineering.

†Professor of Civil Engineering and Computer Science.

‡Group Supervisor, Applied Technologies Section. Member AIAA.

Algorithms for Solution of Differential Matrix Riccati Equation

The differential matrix Riccati equation (RDE) of interest here is the control Riccati equation given as

$$\dot{P}(t) = -P(t)A - A^TP(t) + P(t)BR^{-1}B^TP(t) - Q;$$
$$P(t) = F \tag{1}$$

where $P(t) \in R^{n\times n}$ is the positive semidefinite Riccati matrix, $A \in R^{n\times n}$ and $B \in R^{n\times m}$ are the matrices in the state equation

$$\dot{x}(t) = Ax(t) + Bu(t); \quad 0 \le t \le T \tag{2}$$

and $Q \in R^{n\times n}$ and $R \in R^{m\times m}$ are the positive semidefinite state-weighting matrix and positive definite control-weighting matrix, respectively. In this work, R is assumed to be diagonal and $Q = CC^T$ where $C^T \in R^{p\times n}$ is the output matrix relating state x to sensor output y.

In the following sections, the computational complexities are expressed in terms of the total number of floating point multiplication/division and addition/subtraction operations. Also given are the storage requirements in terms of computer words.

Direct Integration

The direct integration of the RDE from the given terminal condition can be performed by using any of the (R-K) integration methods, multistep methods, predictor-corrector methods with or without step size control. However, the widely used method is the R-K method[10,11,13,19,30,31,33,34] with the fourth-order method being more common.[10,19,30,33] If we change variables $\tau = T - t$ and define $P'(\tau) = P(t)$, Eq. (1) becomes

$$\dot{P}'(\tau) = A^TP'(\tau) + P'(\tau)A + Q - P'(\tau)BR^{-1}B^TP'(\tau);$$
$$P'(0) = F \tag{3}$$

where $\dot{P}'(\tau) = \mathrm{d}/\mathrm{d}\tau\ [P'(\tau)]$. Also assume that the control interval T is divided into s equally spaced segments with step

size $h = T/s$ such that $\tau = ih$, $i = 0,\ldots,s$. Let

$$\mathfrak{F}(P_i') = A^T P_i' + P_i' A + Q - P_i' B R^{-1} B^T P_i' \tag{4}$$

where $P_i' = P'(\tau) = P'(ih)$. The $\mathfrak{F}(\cdot)$ is economically evaluated as

$$\mathfrak{F}(P_i') = (P_i' A)^T + (P_i' A) + Q - (P_i' B) R^{-1} (P_i' B)^T \tag{5}$$

where $P_i' A$ and $P_i' B$ are computed first and then the rest of the computation uses symmetry.

Generalization for the k*th-Order Runge-Kutta Method*

The kth-order R-K method requires k $\mathfrak{F}(\cdot)$ evaluations for $k \leq 4$. For larger k, more than k function evaluations may be needed depending on the particular method used and whether local error estimates are required. Therefore, the asymptotic estimate of complexity for any R-K method that requires β $\mathfrak{F}(\cdot)$ evaluations is $O(\beta s n^3)$. For $k \leq 4$, $\beta = k$. The detailed complexity estimate will depend on the particular method used and whether local error estimates are required. The detailed estimate of complexity is given in line 1 of Table 1, where α, α' are parameters that depend on the particular R-K method chosen. Note that the storage given on this line corresponds to $k = 4$. For k greater/lesser than 4, the storage requirement is also greater/lesser as is to be expected. For example, for $k = 1$, the Euler method, the storage needed is only $3n^2 + 2mn + 2n + m$.

The foregoing estimate is given assuming that the matrix A does not have any special structure. There are cases in which the matrices are simple enough that exploiting such a structure results in savings in computation and storage. Examples are the observer, observability, controller, and controllability canonical forms, where the A matrix is in companion form. Also, the A matrices appearing from FEM models of actively controlled structural systems usually have a small bandwidth. More details on how such advantages in computation and storage can be realized are found in Ref. 50.

Chandrasekhar Algorithm

The Chandrasekhar algorithm[14-17,33,34] transforms the RDE into a set of differential equations for the the gain matrix $K(T - \tau) = K'(\tau)$ given by

$$K'(\tau) = R^{-1} B^T P'(\tau) \tag{6}$$

Here, we assume without loss of generality a zero terminal condition for the RDE. The L-K system of differential equations is then given by

$$\dot{K}'(\tau) = R^{-1} B^T L(\tau) L^T(\tau); \quad K'(0) = 0 \tag{7}$$

$$\dot{L}(\tau) = [A - BK'(\tau)]^T L(\tau); \quad L(0) = C \tag{8}$$

$$0 \leq \tau \leq T$$

where $L \in R^{n \times p}$ and $K \in R^{m \times n}$. L is the square root of the time derivative of the Riccati matrix, that is, $\dot{P}'(\tau) = L(\tau) L^T(\tau)$, which can be obtained by comparing Eqs. (6) and (7). Thus, the total number of differential equations to be integrated is reduced from $n(n + 1)/2$ for the RDE to $n(m + p)$ in this method. Also, the gain matrix is obtained directly, whereas in the case of direct integration of RDE, only the Riccati matrix is obtained and the gain matrix computation requires an additional $n^2 m + nm$ multiplications and $n^2 m$ additions. Thus, for $2(m + p) < (n + 1)$, the number of differential equations here is lesser.

For nonzero initial conditions, a full-rank factorization of $\mathfrak{F}(F)$ has to be performed before the algorithm can be started.[16] The rank of the factored matrix then decides the number of columns in L, which decides the complexity of the algorithm. Brammer[18] has shown that the rank of $\mathfrak{F}(F)$ is in nearly all cases n for a nonzero F for controllable and observable systems. Therefore, for nonzero terminal conditions, the Chandrasekhar algorithm may not be computationally effective. Lainiotis[10-12] and Womble and Potter[13] have generalized the Chandrasekhar algorithm to handle nonzero terminal conditions and also the case of time-varying systems.

Define

$$\mathfrak{F}'(\cdot) = R^{-1} B^T L(\tau) L^T(\tau); \quad \mathfrak{F}''(\cdot) = [A - BK'(\tau)]^T L(\tau)$$

Generalization for the k*th-Order Runge-Kutta Method*

Again, as stated earlier for the direct integration of the RDE, there are β $\mathfrak{F}'(\cdot)$, $\mathfrak{F}''(\cdot)$ evaluations for kth-order R-K method for Eqs. (7) and (8). The detailed complexity for this algorithm is given in line 2 of Table 1. Note that the storage on this line corresponds to $k = 4$. The asymptotic complexity of this method is $O(\beta s n^2 p)$ for $p, m \ll n$, where s is the number of time stations at which the gain matrix is computed.

Just as in the case of direct integration of Riccati equation, savings in computation and storage can be obtained by exploiting the sparsity of the matrices involved, in special situations like the controller, controllability, observer, observability formulations, and structural systems with small bandwidth. For more details, see Ref. 50.

Automatic Synthesis Program

Kalman and Englar[4] proposed the automatic synthesis program (ASP) matrix iteration procedure for solving the RDE. The recursion for the Riccati solution $P[T - (i + 1)h] = P'[(i + 1)h]$ is given as

$$P'[(i + 1)h] = [\phi_{21}(h) + \phi_{22}(h) P'(ih)][\phi_{11}(h) + \phi_{12}(h) P'(ih)]^{-1} \tag{9a}$$

$$P'(0) = F; \quad i = 0,\ldots, s - 1 \tag{9b}$$

where

$$\Phi(h) = e^{Zh} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \tag{10}$$

and where Z is the Hamiltonian matrix given as

$$Z = \begin{bmatrix} -A & BR^{-1}B^T \\ Q & A^T \end{bmatrix} \tag{11}$$

and h is the step size such that the control interval T is divided equally into s intervals so that $s = T/h$. The transition matrix $\Phi(t)$ of the Hamiltonian matrix can be computed in a number of different ways,[54] but in this work, it is assumed to be computed using a Pade second-order approximation.

$$\Phi(h) = e^{Z(h)} \approx [I - hZ/2 + h^2 Z^2/12]^{-1} \times [I + hZ/2 + h^2 Z^2/12] \tag{12}$$

The asymptotic complexity for the ASP is $O[(44/3 + 10s/3)n^3]$. Line 3 of Table 1 gives detailed expressions for the computational complexity and storage requirement for this algorithm. The details are found in Ref. 50.

Davison-Maki Finite-Time Algorithm and Its Modification

Davison and Maki[31] proposed the finite-time algorithm based on the transition matrix of the Hamiltonian matrix Z as in Eq. (11). Assuming $P(T) = 0$, they derived the relation

$$P(t) = [\phi_{21}(-t) + \phi_{22}(-t) P(0)] \times [\phi_{11}(-t) + \phi_{12}(-t) P(0)]^{-1}; \quad 0 \leq t \leq T \tag{13}$$

where

$$P(0) = -\phi_{22}^{-1}(-T) \phi_{21}(-T) \tag{14}$$

and $\Phi(\cdot)$ has the same meaning as in Eq. (12). They introduced the technique to doubling, so that the algorithm becomes much faster due to the lesser number of evaluations of Eq. (13). The doubling technique doubles the time interval of the current one, to be used as the succeeding time interval. The algorithm follows.

1) Fix step size h as $T = 2^{s'}h$ so that the truncation error in the approximation of Eq. (12), $O(h^5)$, is minimized.

2) Compute $\Phi(-h)$ using the second-order Pade approximation of Eq. (12) for the matrix exponential.

3) Make s' matrix multiplications to find and store $e^{Z(-2h)} = \Phi \cdot \Phi, \ldots, e^{Z(-2^ih)} = \Phi^{2^{i-1}} \cdot \Phi^{2^{i-1}} = \Phi^{2^i}, \ldots$, while scaling at each step by the ∞ norm of the matrices to avoid overflow.

4) Compute $P(0)$ using Eq. (14).

5) Compute $P(2^ih)$ using the stored transition matrices and Eq. (13) for $i = 0, \ldots, s' - 1$.

The asymptotic complexity for this algorithm is $O[(16 + 34s'/3)n^3]$ for large n. The detailed expressions for storage and complexity are given in line 4 of Table 1. The most disadvantageous feature of this algorithm is the storage expressed asymptotically as $O[(4s' + 4)n^2]$. The authors suggest typical values of s' between 5 and 20. This makes the algorithm impractical for large-order systems.

Here, we alleviate the storage problem by modifying the algorithm in the following manner. We start the solution for $P'(\tau)$, when $P'(0) = F = \mathbf{0}$, as

$$P'(\tau) = \phi_{21}(\tau)\phi_{11}^{-1}(\tau); \quad 0 \le \tau \le T \tag{15}$$

where $P'(\tau)$ is the Riccati solution $P(T - \tau)$. Again, if we use the doubling technique $\tau = 2^ih$, Eq. (15) can be rewritten as

$$P'(2^ih) = \frac{\phi_{21}(2^ih)}{||\Phi(2^ih)||_\infty}\left[\frac{\phi_{11}(2^ih)}{||\Phi(2^ih)||_\infty}\right]^{-1}; \quad i = 0, \ldots, s \tag{16}$$

where $||\Phi(2^ih)||_\infty$ is the infinity norm of $\Phi(2^ih)$. This scaling is done to avoid overflow. $\Phi(2^ih)$ is computed as $\Phi(2^{i-1}h) \cdot \Phi(2^{i-1}h)$. The asymptotic storage for this modified method is only $O(8n^2)$ and its complexity asymptotically reaches $O[(44/3 + 28s'/3)n^3]$. Thus, there is a large saving in storage space because the necessity to store all the transition matrices has been removed. Also, this method is asymptotically 20% faster than the Davison–Maki algorithm. Also, the transient solution is better represented because of the greater number of time stations on the transient portion of the interval than in the Davison-Maki algorithm. The details of this modification are given in Ref. 50. The detailed complexity and storage expressions are given in line 5 of Table 1.

Vaughan's Negative Exponential Solution

Vaughan[8] proposed the solution to the RDE in terms of the negative exponential of the right half-plane eigenvalues of the Hamiltonian matrix and the corresponding eigenvectors matrix. Let Z in Eq. (11) be diagonalized as $Z = WDW^{-1}$, where

$$D = \begin{bmatrix} \Lambda & \mathbf{0} \\ \mathbf{0} & -\Lambda \end{bmatrix}$$

is the matrix of eigenvalues of Z and

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}$$

is the eigenvector matrix of Z and Λ is a diagonal matrix having all right half-plane eigenvalues on its main diagonal. If Λ is complex, W has to be complex and the computation becomes expensive. This can be avoided by using a complex block-diagonal similarity transformation so that D becomes block diagonal with 2×2 blocks for every complex-conjugate eigenvalue pair on the diagonal. The complex-conjugate pair of eigenvectors also become a pair of real vectors due to this transformation. This was used by Fath[28] in his eigenvector algorithm for the algebraic matrix Riccati equation and is also mentioned in Wilkinson.[51] The Riccati solution $P(T - \tau) = P'(\tau)$ at time $t = T - \tau$ is given as

$$P'(\tau) = [W_{21} + W_{22}G(\tau)][W_{11} + W_{12}G(\tau)]^{-1}; \quad 0 \le \tau \le T \tag{17}$$

where

$$G(\tau) = e^{-\Lambda\tau}Re^{-\Lambda\tau} \tag{18}$$

$$R = -[W_{22} - FW_{12}]^{-1}[W_{21} - FW_{11}] \tag{19}$$

and where F is the terminal condition to the Riccati equation (1). This algorithm first requires an eigenvalue-eigenvector analysis of the Hamiltonian matrix $\dot{Z}$. For our analysis, we assume that the matrices W and D are computed using Fath's algorithm.[28] The steps in Vaughan's method are the following:

1) Compute W and D using Fath's eigenvector algorithm.
2) Compute R from Eq. (19).
3) For $i = 0, \ldots, s$, where $s = T/h$, do
4) Compute $G(ih) = (e^{-\Lambda h})^i R (e^{-\Lambda h})^i$.
5) Compute $P'(ih)$ from Eq. (17).
6) Stop.

Steps 1 and 4 have varying complexities depending on the nature of the spectrum of Z. The complexities for these steps are given for the two extreme cases of all real and all complex-conjugate pairs of eigenvalues. The detailed estimates for the complexity for each of the steps in the algorithm are given in Ref. 50. The total complexity and storage requirement are given in lines 6(a) and (b) of Table 1. The asymptotic complexity varies in the range of $O[(70 + 10s/3)n^3]$ to $O[(77 + 10s/3)n^3]$. The asymptotic storage requirement is $O(16.5n^2)$ for large n.

Continuous Square-Root Algorithms

Continuous square-root algorithms[17,26,37,38] give differential equation for the square root of the Riccati matrix. Since the Riccati matrix P is symmetric and positive definite for controllable and observable systems, we are guaranteed that a nonsingular matrix S exists such that $P = SS^T$[53]. In this work, we analyze the eigenfactor algorithm proposed by Oshman et al.,[26] and the square-root algorithm by Morf et al.[17]

Morf-Levy-Kailath Square-Root Algorithm

Morf et al.[17] derived the following differential equation for the Cholesky factor S of the positive definite Riccati matrix P:

$$\dot{S}(\tau) = S(\tau)[M(\tau)]_{+/2}; \quad S(0) = F_0; \quad F_0F_0^T = F \tag{20}$$

$$M(\tau) = S^{-1}A^TS + S^TAS^{-T} + S^{-1}CC^TS^{-1} - S^TBR^{-1}B^TS \tag{21}$$

where $[\cdot]_{+/2}$ is the lower-triangular operator defined by

$$[M]_{+/2ij} = \begin{cases} 0; & i < j \\ (m_{ij}/2); & i = j \\ m_{ij}; & i > j \end{cases}$$

The Riccati solution $P(T - \tau) = P'(\tau)$ is then obtained as $P'(\tau) = S(\tau)S^T(\tau)$. The reader is referred to Ref. 17 for more details.

If we assume a kth-order R-K integration procedure for the integration of Eq. (20) that requires β function evaluations for every time point, the asymptotic complexity of this algorithm is $O[(7/6)s\beta n^3]$. The storage requirement and complexity in detail are given in line 7 of Table 1.

Eigenfactor Algorithm

Oshman et al.[26] introduced the eigenfactor square-root algorithm for the RDE by giving differential equations to the

Table 1 Computational complexities and storage requirements for the solution algorithms of the differential matrix Riccati equation in terms of the plant parameters n, m, p, and parameters of the numerical method s, s', s'', q, q', β, α, α'

No.	Algorithm		Computational complexity: Detailed	Computational complexity: Asymptotic	Storage requirement: Detailed	Storage requirement: Asymptotic
1	R-K direct integration	M/D	$sn^3\beta + sn^2(1.5\beta m + 0.5\beta + 0.5\alpha) + sn(2.5\beta m + 0.5\beta + 0.5\alpha) - s\beta m$	$O(sn^3\beta)$	$5n^2 + m + 2mn + 4n$	$O(5n^2)$
		A/S	$sn^3\beta + sn^2(1.5\beta m + 2\beta + 0.5\alpha') + sn(0.5\beta m + 2\beta + 0.5\alpha')$	$O(sn^3\beta)$		
2	Chandrasekhar*	M/D	$sn^2\beta p + sn(3\beta mp + \beta p + \alpha m + \alpha p) + 3s\beta mp$	$O(sn^2\beta p)$	$n^2 + mp + m$	$O(n^2)$
		A/S	$sn^2\beta p + sn(3\beta mp + \beta p + \alpha' m + \alpha' p + \beta m)$	$O(sn^2\beta p)$	$6nm + 5np$	
3	ASP	M/D	$n^3(44/3 + 10s/3) + n^2(8 + 0.5m) + n(1.5m - s/3 - 2/3)$	$O[n^3(44 + 10s)/3]$	$8.5n^2 + 1.5n$	$O(8.5n^2)$
		A/S	$n^3(44/3 + 10s/3) + n^2(4 + 0.5m + 0.5s) + n(0.5m + s/6 + 1/3)$	$O[n^3(44 + 10s)/3]$		
4	Davison-Maki algorithm	M/D	$n^3(16 + 34s'/3) + n^2(0.5m + 4s' + 8) + n(1.5m - s'/3 - 1)$	$O[n^3(16 + 34s'/3)]$	$(4s' + 4)n^2$	$O[(4s' + 4n^2)]$
		A/S	$n^3(16 + 34s'/3) + n^2(0.5m + 4.5s' + 2.5) + n(0.5m + s'/6 + 1/6)$	$O[n^3(16 + 34s'/3)]$		
5	Modified Davison-Maki	M/D	$n^3(44/3 + 28s'/3) + n^2(0.5m + 4.5s' + 2.5) + n(1.5m - s'/3)$	$O[n^3(44 + 28s')/3]$	$8n^2$	$O(8n^2)$
		A/S	$n^3(44/3 + 28s'/3) + n^2(1.5s' + 0.5m + 9.5) + n(0.5m - 7/3)$	$O[n^3(44 + 28s')/3]$		
6	Negative exponential Solution	M/D	$n^3(86/3 + 10s/3 + 20q + 4q') + n^2(16s'' + 42q + 14q' - 4) - n(32/3 + 10s'' + 58q + 2q' + s/3) - (4q + 7 - 2s'')$	$O[n^3(86/3 + 4q' + 20q + 10s/3)]$	$16.5n^2 + 6.5n$	$O(16.5n^2)$
(a)	All real eigenvalues	A/S	$n^3(86/3 + 4q' + 24q + 10s/3) + n^2(26q + q' - 3s/2 - 21/2) - n(41q + q' - s/6 + 25/6) - (7 + 9q)$	$O[n^3(86/3 + 4q' + 24q + 10s/3)]$		
		$\sqrt{\cdot}$	$n^2(2q) + n(2 - q) - (q + 2)$	$O(2qn^2)$		
(b)	All complex eigenvalues	M/D	$n^3(98/3 + 8q' + 10q + 10s/3) + n^2(16s'' + 42q + 14q' + 2s - 4) - n(32/3 + 10s'' + 58q + 2q' + s/3) - (4q + 7 - 2s'')$	$O[n^3(98/3 + 8q' + 10q + 10s/3)]$	$16.5n^2 + 6.5n$	$O(16.5n^2)$
		A/S	$n^3(98/3 + 8q' + 12q + 10s/3) + n^2(19q + 14q' - 3s/2 - 15/2) - n(37/6 + 15q + 6q' - s/6) - (16q + 7)$	$O[n^3(98/3 + 8q' + 12q + 10s/3)]$		
		$\sqrt{\cdot}$	$n^2(q) + 2n - (q + 2)$	$O(qn^2)$		
7	Morf et al. square-root* algorithm	M/D	$sn^3\left(\frac{7\beta}{6}\right) + sn^2\left(\frac{3\beta p}{2} + \frac{3\beta m}{2} + 2\beta + \frac{\alpha}{2}\right) + sn\left(\frac{3\beta m}{2} + \frac{\beta p}{2} + \frac{5\beta}{6} + \frac{\alpha}{2}\right)$	$O\left(\frac{7\beta}{6}sn^3\right)$	$4.5n^2 + 2.5n + m + 2mn + 2np$	$O(4.5n^2)$
		A/S	$sn^3\left(\frac{7\beta}{6}\right) + sn^2\left(\frac{3\beta p}{2} + \frac{3\beta m}{2} + \frac{3\beta}{2} + \frac{\alpha'}{2}\right) + sn\left(\frac{\beta m}{2} - \frac{\beta p}{2} + \frac{4\beta}{3} + \frac{\alpha'}{2}\right)$	$O\left(\frac{7\beta}{6}sn^3\right)$		
8	Eigenfactor algorithm	M/D	$sn^4\left(\frac{5\beta}{4}\right) + sn^3\left(\frac{5\beta + 1}{2}\right) + sn^2\left(\frac{11\beta}{4} + \frac{3}{2} + \alpha\right) + sn\alpha$	$O\left(\frac{5\beta}{4}sn^4\right)$	$9n^2 + mn + 8n + m$	$O(9n^2)$
		A/S	$sn^4\left(\frac{7\beta}{2}\right) + sn^3\left(2.0\beta + \frac{1}{2}\right) + sn^2\left(\frac{13\beta}{4} + \alpha' + \frac{1}{2}\right) + sn\alpha'$	$O\left(\frac{7\beta}{2}sn^4\right)$		

eigenvalues and eigenvectors of the positive semidefinite symmetric Riccati matrix. They propose the solution to Eq. (3) as

$$\boldsymbol{P}'(\tau) = \boldsymbol{V}(\tau)\boldsymbol{\Lambda}(\tau)\boldsymbol{V}^T(\tau) \tag{22}$$

where

$$\dot{\boldsymbol{\Lambda}}(\tau) = \boldsymbol{\Gamma}(\tau); \quad \boldsymbol{\Lambda}(0) = \boldsymbol{\Lambda}_0 \tag{23}$$

$$\dot{\boldsymbol{V}}(\tau) = \boldsymbol{V}(\tau)\boldsymbol{\Omega}(\tau); \quad \boldsymbol{V}(0) = \boldsymbol{V}_0 \tag{24}$$

$$\boldsymbol{\Gamma}(\tau) = \text{diag}[\gamma_{ii}(\tau)] \tag{25}$$

$$\gamma_{ji}(\tau) = \gamma_{ij}(\tau) = \boldsymbol{v}_j^T(\tau)\boldsymbol{T}_{ij}(\tau)\boldsymbol{v}_i(\tau) \tag{26}$$

$$\boldsymbol{T}_{ji}^T(\tau) = \boldsymbol{T}_{ij}(\tau) = [\lambda_i(\tau)\boldsymbol{A}^T + \lambda_j(\tau)\boldsymbol{A} + \boldsymbol{Q} - \lambda_i(\tau)\lambda_j(\tau)\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T] \tag{27}$$

$$\omega_{ij}(\tau) = \begin{cases} \dfrac{\gamma_{ji}(\tau)}{\lambda_j(\tau) - \lambda_i(\tau)}; & \left|\dfrac{\gamma_{ji}(\tau)}{\lambda_j(\tau) - \lambda_i(\tau)}\right| < \omega_{\max} \\ 0; & \lambda_i(\tau) = \lambda_j(\tau) \\ \omega_{\max}\ \text{sgn}\left[\dfrac{\gamma_{ji}(\tau)}{\lambda_j(\tau) - \lambda_i(\tau)}\right]; & \left|\dfrac{\gamma_{ji}(\tau)}{\lambda_j(\tau) - \lambda_i(\tau)}\right| \geq \omega_{\max} \end{cases} \tag{28}$$

where $\boldsymbol{P}'(\tau)$ is the Riccati solution $\boldsymbol{P}(T - \tau)$. $\boldsymbol{\Lambda}_0$ and $\boldsymbol{V}_0$ are obtained by diagonalizing the terminal condition $\boldsymbol{F}$. The

Table 2 Computational complexities and storage requirements for the solution algorithms of the algebraic matrix Riccati equation in terms of the plant parameters n, m and parameters of the numerical method s'', q, q', β, k_d, k_s, $k_{\max}$, $k'_{\max}$, $\Delta\epsilon$, $\mathfrak{L}_i(n)$, $\mathfrak{M}_i(n)$

No.	Algorithm		Comutational complexity: Detailed	Comutational complexity: Asymptotic	Storage requirement: Detailed	Storage requirement: Asymptotic
1	Newton	M/D	$k_{\max}[2.5n^2m + 1.5mn + \mathfrak{L}_i(n)]$	$k_{\max}O[\mathfrak{L}_i(n)]$	$1.5n^2 + 2mn + m + 0.5n + \mathfrak{M}_i(n)$	$1.5n^2 + O[\mathfrak{M}_i(n)]$
		A/S	$k_{\max}[2.5n^2m + 0.5mn + \mathfrak{L}_i(n)]$	$k_{\max}O[\mathfrak{L}_i(n)]$		
2	Q-Imbedded	M/D	$[2n^2m + mn + \beta\mathfrak{L}_i(n)]/\Delta\epsilon$	$\beta O[\mathfrak{L}_i(n)]/\Delta\epsilon$	$2.5n^2 + 2mn + 0.5n + m + \mathfrak{M}_i(n)$	$2.5n^2 + O[\mathfrak{M}_i(n)]$
		A/S	$[2n^2m + mn + \beta\mathfrak{L}_i(n)]/\Delta\epsilon$	$\beta O[\mathfrak{L}_i(n)]/\Delta\epsilon$		
3	Combined imbedded-Newton	M/D	$4n^2m + 2mn + (2\beta + k'_{\max})\mathfrak{L}_i(n)$	$(2\beta + k'_{\max})O[\mathfrak{L}_i(n)]$	$2.5n^2 + 2mn + 0.5n + m + \mathfrak{M}_i(n)$	$2.5n^2 + O[\mathfrak{M}_i(n)]$
		A/S	$4n^2m + 2mn + (2\beta + k'_{\max})\mathfrak{L}_i(n)$	$(2\beta + k_{\max})O$		
4	Eigenvector method	M/D	$n^3(62/3 + 20q + 2q') + n^2(16s'' + 42q + 3q' + m - 1) - n(58q + 10s'' - m + q' + 23/3) + (2s'' - 4q - 7)$	$O[n^3(20q + 2q' + 62/3)]$	$14n^2 + 6n$	$O(14n^2)$
	(a) All eigenvalues real	A/S	$n^3(62/3 + 24q + 2q') + n^2(26q + 8s'' + q' + m - 17/2) - n(41q + 4s'' + q' + 4/3) - (9q + 7)$	$O[n^3(24q + 2q' + 62/3)]$		
		$\sqrt{\cdot}$	$2qn^2 + n(2 - q) - (2 + q)$	$O(2qn^2)$		
	(b) All eigenvalues complex	M/D	$n^3(68/3 + 10q + 4q') + n^2(16s'' + 26q + 12q' + m) - n(38q + 4q' + 10s'' - m - 26/3) + (2s'' - 17q - 7)$	$O[n^3(10q + 4q' + 68/3)]$	$14n^2 + 6n$	$O(14n^2)$
		A/S	$n^3(68/3 + 12q + 4q') + n^2(19q + 7q' + 8s'' + m - 15/2) - n(15q + 3q' + 4s'' + 7/3) - (7 + 16q)$	$O[n^3(4q' + 12q + 68/3)]$		
		$\sqrt{\cdot}$	$qn^2 + 2n - (2 + q)$	$O(qn^2)$		
5	Schur method	M/D	$n^3(20 + 89q/2) + n^2(67q/2 + m - 6) - n(55q - m + 6) - (16 + 4q)$	$O[n^3(89q/2 + 20)]$	$8n^2 + 2n$	$O(8n^2)$
	(a) Schur form triangular	A/S	$n^3(20 + 54q) + n^2(16q + m - 39/2) + n(34/3 - 41q) - (14 + 9q)$	$O[n^3(54q + 20)]$		
		$\sqrt{\cdot}$	$2.5qn^2 + (2 - 0.5q)n - (2 + q)$	$O(2.5qn^2)$		
	(b) Schur form has all 2×2 bumps	M/D	$n^3(20 + 53q/2) + n^2(365q/8 + m - 6) - n(15q/4 + 6 - m) - (16 + 17q)$	$O[n^3(53q/2 + 20)]$	$8n^2 + 2n$	$O(8n^2)$
		A/S	$n^3(20 + 31q) + n^2(151q/4 + m - 39/2) + n(34/3 - 11q/2) - (14 + 16q)$	$O[n^3(31q + 20)]$		
		$\sqrt{\cdot}$	$11qn^2/8 + n(2 + 3q/4) - (2 + q)$	$O(11qn^2/8)$		
6	Matrix sign function	M/D	$(k_d + 1)\left[n^3\left(\frac{20}{3}k_s + \frac{19}{3}\right) + n^2\left(2k_s - \frac{3}{2}\right) + n\left(\frac{10}{3}k_s + \frac{5}{3}\right)\right]$	$O\left[n^3(k_d + 1)\left(\frac{20}{3}k_s + \frac{19}{3}\right)\right]$	$10.5n^2 + 6.5n$	$O(10.5n^2)$
		A/S	$(k_d + 1)\left[n^3\left(\frac{20}{3}k_s + \frac{19}{3}\right) + n^2(2k_s) + n\left(\frac{4}{3}k_s + \frac{8}{3}\right)\right]$	$O\left[n^3(k_d + 1)\left(\frac{20}{3}k_s + \frac{19}{3}\right)\right]$		

integration of Eqs. (23) and (24) can be done by R-K methods or any other suitable initial value method. If we assume a kth-order R-K integration procedure with β function evaluations, the asymptotic complexity for the continuous square-root algorithm becomes $O(1.25\beta sn^4)$, where s is the number of time stations at which the Riccati solution is computed. The storage requirement and complexity in detail are given in line 8 of Table 1. Note that the storage requirement on this line corresponds to $k = 4$. For more details on how these expressions are arrived at, see Ref. 50.

Algorithms for Solution of Algebraic Matrix Riccati Equation

It is well known[52] that if the system is stabilizable and detectable, the differential matrix Riccati equation solution $P(t)$ reaches a steady state for large control intervals. In this case, the differential equation degenerates to the algebraic Riccati equation (ARE)

$$PA + A^TP + PBR^{-1}B^TP - Q = 0 \tag{29}$$

There are various algorithms in the literature to obtain the solution of the ARE. All the methods discussed in this section also can be used to obtain the steady-state solution, by integrating up to the time when a steady-state solution is reached, that is, the $P(t)$ does not change by some small norm value. However, it will be shown later that for large systems, these methods would not be as efficient as the algorithms meant especially for the ARE. These algorithms are discussed next.

Newton-Raphson Method

When the Newton method is applied to Eq. (29), one may obtain the iteration

$$A_k^TP_{k+1} + P_{k+1}A_k = -Q_k; \quad k = 0,... \tag{30}$$

where $A_k = A - BR^{-1}B^TP_k$ and $Q_k = P_kBR^{-1}B^TP_k + Q$ where P_0 is chosen such that it makes A_0 stable. It has been shown[6] that such a A_0 leads to a sequence of converging iterates P_k, $k = 0,...$ with each P_k being positive semidefinite. Also the convergence has been shown to be quadratic.[6]

Table 3 Computational complexities and storage requirements for the solution algorithms of the Lyapunov equation in terms of the plant parameter n and parameters of the numerical method k_1, k_2, q

No.	Algorithm		Computational complexity $\mathfrak{L}_i(n)$ Detailed	Computational complexity $\mathfrak{L}_i(n)$ Asymptotic	Storage requirement $\mathfrak{M}_i(n)$ Detailed	Storage requirement $\mathfrak{M}_i(n)$ Asymptotic
1	Lyapunov solver 1	M/D	$n^3(2.5k_1 + 7/3) + n^2(0.5k_1 + 2) - 4n/3$	$O[n^3(2.5k_1 + 7/2)]$	$3.5n^2 + 1.5n$	$O(3.5n^2)$
		A/S	$n^3(2.5k_1 + 7/3) + n^2(k_1 + 4) + n(0.5k_1 + 16/3)$	$O[n^3(2.5k_1 + 7/3)]$		
2	Lyapunov solver 2	M/D	$n^3(2.5k_1 + 4/3) + n^2(0.5k_1 + 2.5) + n/6$	$O[n^3(2.5k_1 + 4/3)]$	$3.5n^2 + 1.5n$	$O(3.5n^2)$
		A/S	$n^3(2.5k_1 + 4/3) + n^2(k_1 + 4) + n(0.5k_1 + 2/3)$	$O[n^3(2.5k_1 + 4/3)]$		
3	Lyapunov solver 3	M/D	$n^3(17k_2/6) + n^2(2.5k_2 + 0.5) + n(2k_2/3 + 0.5) + 6k_2$	$O(17k_2n^3/6)$	$3.5n^2 + 1.5n$	$O(3.5n^2)$
		A/S	$n^3(17k_2/6) + n^2(0.5k_2 + 0.5) + n(2k_2/3 + 0.5)$	$O(17k_2n^3/6)$		
4	Lyapunov solver 4	M/D	$n^3(5q + 35/6) + n^2(6q + 1) - n(30q + 17/6) - (16 + 4q)$	$O[n^3(35/6 + 5q)]$	$2.5n^2 + 3.5n$	$O(2.5n^2)$
	(a) Schur form triangular	A/S	$n^3(6q + 35/6) - n^2(5 - 3q)/2 + n(11/3 - 45q/2) - (14 + 9q)$	$O[n^3(35/6 + 6q)]$		
		$\sqrt{\cdot}$	$n^2q/2 + n(1 - q/2) - (2 + q)$	$O(qn^2/2)$		
	(b) Schur form has all 2×2 bumps on diagonal	M/D	$n^3(35/6 + 5q/2) + n^2(3 + 11q/2) - n(10/3 + 25q/2) - (17q + 16)$	$O[n^3(35/6 + 5q/2)]$	$2.5n^2 + 3.5n$	$O(2.5n^2)$
		A/S	$n^3(73/12 + 3q) + n^2(15q/4 - 1) + n(29/3 - 23q/2) - (14 + 16q)$	$O[n^3(73/12 + 3q)]$		
		$\sqrt{\cdot}$	$n^2q/4 + n - (2 + q)$	$O(qn^2/4)$		
5	Lyapunov solver 5	M/D	$n^3(5q + 35/6) + n^2(6q + 6) - n(30q + 11/6) - (16 + 4q)$	$O[n^3(35/6 + 5q)]$	$3.5n^2 + 3.5n$	$O(3.5n^2)$
	(a) Schur form triangular	A/S	$n^3(6q + 35/6) + n^2(7 + 3q)/2 + n(11/3 - 45q/2) - (14 + 9q)$	$O[n^3(35/6 + 6q)]$		
		$\sqrt{\cdot}$	$n^2(q + 1)/2 + n(5 - q)/2 - (2 + q)$	$O[n^2(q + 1)/2]$		
	(b) Schur form has all 2×2 bumps on diagonal	M/D	$n^3(11/2 + 5q/2) + n^2(12 + 11q/2) + n(97 - 25q/2) - (17q + 16)$	$O[n^3(11/2 + 5q/2)]$	$3.5n^2 + 3.5n$	$O(3.5n^2)$
		A/S	$n^3(67/12 + 3q) + n^2(15q + 15)/4 + n(95 - 23q/2) - (14 + 16q)$	$O[n^3(67/12 + 3q)]$		
		$\sqrt{\cdot}$	$n^2(1/2 + q/4) + 9n/2 - (2 + q)$	$O[n^2(1/2 + q/4)]$		

Equation (30) is the Lyapunov equation. Since every iteration of Newton's method requires the solution of a Lyapunov equation, the computational complexity and storage requirement for this method are expressed in terms of the computational complexity and storage requirement for each of the five Lyapunov equation solvers considered. Thus, if $\mathfrak{L}_i(n)$ and $\mathfrak{M}_i(n)$ are, respectively, the complexity and storage requirement for each of the Lyapunov equation solvers $i = 1, \ldots,5$, the complexity and storage requirement of the Newton method using each of these Lyapunov solvers are given in terms of $\mathfrak{L}_i(n)$ and $\mathfrak{M}_i(n)$ in line 1 of Table 2. Table 3 contains the detailed estimate of $\mathfrak{L}_i(n)$ and $\mathfrak{M}_i(n)$ for $i = 1, \ldots,5$.

The computational complexity of the Newton method is given by $k_{\max}[2.5n^2m + mn + \mathfrak{L}_i(n)]$, where $k_{\max}$ is the number of Newton iterations required for convergence. The storage requirement is given by $1.5n^2 + 2mn + 0.5n + m + \mathrm{M}_i(n)$. In the following sections, we discuss briefly five selected methods for solving the Lyapunov equation. There are many more in the literature,[39-43] but all of them require much more computational effort or storage than those described here and some do not have favorable numerical properties. Detailed complexity and storage requirements for these algorithms can be found in Ref. 50. For notational convenience, we drop the subscripts in Eq. (30) and consider the Lyapunov equation as

$$\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} = -\boldsymbol{C} \tag{31}$$

Lyapunov Solver 1

R. A. Smith[44] proposed the following recursive algorithm to solve the Lyapunov equation. Let q be a positive parameter and define

$$\boldsymbol{U} = (q\boldsymbol{I} - \boldsymbol{A}^T)^{-1}; \quad \boldsymbol{V} = \boldsymbol{U}(q\boldsymbol{I} + \boldsymbol{A}^T); \quad \boldsymbol{W} = 2q\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^T$$

Then if $\boldsymbol{A}$ is Hurwitz

$$\boldsymbol{P} = \sum_{i=1}^{\infty} \boldsymbol{V}^{i-1}\boldsymbol{W}(\boldsymbol{V}^{i-1})^T$$

converges and is the solution to the Lyapunov equation. Let $\boldsymbol{P}_0 = \boldsymbol{W}$. Then, the foregoing series is computed efficiently using the recursion

$$\boldsymbol{P}_{i+1} = \boldsymbol{P}_i + (\boldsymbol{V})^{2^i \times 2}\boldsymbol{P}_i(\boldsymbol{V}^T)^{2^i \times 2}; \quad i = 0,\ldots,k_1$$

where k_1 is the number of recursions required to obtain $\boldsymbol{P}$, the solution of the Lyapunov equation, with sufficient accuracy. The complexity $\mathfrak{L}_1(n)$ and storage $\mathfrak{M}_1(n)$ are given in line 1 of Table 3. The asymptotic estimates are $O[(7/3 + 2.5k_1)n^3]$ and $O(3.5n^2)$, respectively, for complexity and storage.

Lyapunov Solver 2

Davison and Man[45] proposed a solution to the Lyapunov equation based on the transition matrix of the system, $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, which involves a recursive relation similar to the one in Smith's algorithm. Let $\boldsymbol{P}_0 = h\boldsymbol{C}$ and

$$\boldsymbol{\Phi} = e^{\boldsymbol{A}h} \approx (\boldsymbol{I} - h\boldsymbol{A}/2 + h^2\boldsymbol{A}^2/12)^{-1}(\boldsymbol{I} + h\boldsymbol{A}/2 + h^2\boldsymbol{A}^2/12) \tag{32}$$

Then

$$\boldsymbol{P}_{i+1} = (\boldsymbol{\Phi}^T)^{2^i}\boldsymbol{P}_i(\boldsymbol{\Phi})^{2^i} + \boldsymbol{P}_i; \quad i = 0,\ldots,k_1$$

converges to the solution $\boldsymbol{P}$ of the Lyapunov equation if $\boldsymbol{A}$ is stable. The small step size h is so chosen that the approximation in Eq. (32) is reasonably accurate. The complexity $\mathfrak{L}_2(n)$ and storage $\mathfrak{M}_2(n)$ are given in line 2 of Table 3. The asymptotic estimates are $O[(4/3 + 2.5k_1)n^3]$ and $O(3.5n^2)$, respectively.

Lyapunov Solver 3

Hoskins et al.[46] proposed an algorithm for the Lyapunov equation as follows:

1) $m = 1$; $C_0 = C$.
2) Estimate spectrum of $A_0 = A$ within some range (b,B).
3) Compute a_1, b_1, e from the relations

$$a_1 = \frac{2b}{b + \sqrt{bB}}; \quad b_1 = b_1 B a_1; \quad e = \frac{(b - \sqrt{bB})^2}{(b + \sqrt{bB})^2};$$

4) Compute $A_m = a_1 A_{m-1} + b_1 (A_{m-1})^{-1}$.
5) Compute $C_m = a_1 C_{m-1} + b_1 A_{m-1}^{-T} C_{m-1} A_{m-1}^{-1}$.
6) The new $b = 1 - e$ and $B = 1 + e$.
7) Repeat steps 3-6 until $A_m \approx I$ at $m = k_2$.
8) The Lyapunov solution $P = -C_{k_2}/2$.

The algorithm converges if A is stable. The complexity $\mathfrak{L}_3(n)$ and storage $\mathfrak{M}_3(n)$ are given in line 3 of Table 3. This algorithm has asymptotic complexity and storage of $O(17k_2n^3/6)$ and $O(3.5n^2)$, respectively.

Lyapunov Solver 4

Bartels and Stewart[47] proposed an algorithm that is acclaimed as the best Lyapunov equation solver available in the literature. The Lyapunov equation is transformed into a simpler equation by use of Schur transformations. The simplified equation requires only the solution of a linear system of equations of order at most four. The main steps in the algorithm are as follows:

1) Reduce A to upper Hessenberg form.
2) Transform the upper Hessenberg form to upper real Schur form using a double QR algorithm.
3) Compute the overall orthogonal transformation matrix that transforms A to upper real Schur form.
4) Transform $-C$ in the Lyapunov equation (31) by the Schur transformation matrix.
5) Solve the transformed Lyapunov equation by the partitioning approach to obtain the transformed Lyapunov solution.
6) Transform the Lyapunov solution obtained in step 5 by the Schur transformation matrix to obtain the Lyapunov solution P of Eq. (31).

The steps 2, 3, and 5 have varying complexities depending on the particular Schur form of A obtained, which depends on the nature of spectrum of A. Thus, there are two extreme cases to consider. Case (a) is the case in which the Schur form has all 2×2 submatrices on the diagonal corresponding to comple-conjugate pairs of eigenvalues. Case (b) is the case in which the Schur form is triangular corresponding to all real eigenvalues. Let the complexity in cases (a) and (b) be denoted by $\mathfrak{L}_4^{(a)}(n)$ and $\mathfrak{L}_4^{(b)}(n)$. The storage $\mathfrak{M}_4(n)$ is the same in both cases. The total complexity and storage in (a) and (b) are given in lines 4 and 5 in Table 3. For details on how these complexities are obtained, see Ref. 50. If we use a typical value for the iterative parameter in the complexity expression $q = 2$, the asymptotic estimate for the complexity varies approximately between $O(11n^3)$ to $O(16n^3)$ for this method. The asymptotic storage is $O(2.5n^2)$. The method also requires square-root evaluations, asymptotically between $O(0.5n^2)$ to $O(n^2)$.

Lyapunov Solver 5

Hammarling[48] proposed a variant of the Bartels-Stewart algorithm, which obtains the Cholesky factors of the nonnegative definite Lyapunov solution P. This algorithm is superior to the Bartels-Stewart algorithm in that it always produces a symmetric nonnegative definite solution to the Lyapunov equation, whereas the Bartels-Stewart algorithm produces a symmetric solution that may or may not be nonnegative definite because of roundoff errors. The algorithm uses the same Schur transformations that the Bartels method does. Let $C = R^T R$ and $P = U^T U$. Then, Eq. (31) becomes

$$A^T(U^T U) + (U^T U)A = -R^T R \tag{33}$$

If A has the real Schur form $S = Q^T A Q$ and RQ has the QU-factorization $RQ = \tilde{P}\tilde{R}$, where $\tilde{P}$ is orthonormal, $\tilde{R}$ is upper-triangular, and if we define $\tilde{U} = UQ$, then Eq. (33) can be rewritten as

$$S^T(\tilde{U}^T\tilde{U}) + (\tilde{U}^T\tilde{U})S = -\tilde{R}^T\tilde{R} \tag{34}$$

Let S, $\tilde{U}$, $\tilde{R}$ have the following structure:

$$S = \begin{bmatrix} s_{11} & s^T \\ & S_1 \end{bmatrix}; \quad \tilde{U} = \begin{bmatrix} \tilde{u}_{11} & \tilde{u}^T \\ & U_1 \end{bmatrix}; \quad \tilde{R} = \begin{bmatrix} \tilde{r}_{11} & \tilde{r}^T \\ & R_1 \end{bmatrix}$$

Then, Eq. (36) can be broken into the following three equations for the partitions in the matrices just shown:

$$s_{11}^T(\tilde{u}_{11}^T\tilde{u}_{11}) + (\tilde{u}_{11}^T\tilde{u}_{11})s_{11} = \tilde{r}_{11}^T r_{11} \tag{35}$$

$$S_1^T\tilde{u} + \tilde{u}(\tilde{u}_{11}s_{11}\tilde{u}_{11}^{-1}) = -(\tilde{r}\alpha + s\tilde{u}_{11}^T) \tag{36}$$

$$S_1^T(U_1^T U_1) + (U_1^T U_1)S_1 = -\hat{R}^T\hat{R} \tag{37}$$

where

$$\hat{R}^T\hat{R} = R_1^T R_1 + yy^T, \quad y = \tilde{r} - \tilde{u}\alpha^T, \quad \alpha = \tilde{r}_{11}^T\tilde{u}_{11}^{-1} \tag{38}$$

The block s_{11} can be a scalar or a 2×2 matrix depending on the eigenvalue(s) associated with that block. Equation (37) is an equation similar to Eq. (34), but of a smaller size. Computation of $\hat{R}$ requires Cholesky updating that can be done by Givens rotations. The main steps in the algorithm are as follows:

1) Decompose C into Cholesky factors R, R^T.
2) Transform A to real Schur form S and accumulate the Schur transformations in Q.
3) Perform a QU-factorization of RQ to obtain the triangular factor $\tilde{R}$.
4) Compute the Cholesky factor $\tilde{U}$ by using the partitioning technique signified by Eqs. (35-38).
5) Perform QU-factorization of $\tilde{U}Q^T$ to obtain the triangular factor U.
6) Compute Lyapunov solution $P = U^T U$.

The detailed complexity $\mathfrak{L}_5(n)$ and storage $\mathfrak{M}_5(n)$ required are given in lines 6 and 7 of Table 3. If we use a typical value of the iterative parameter in the complexity expression $q = 2$, the asymptotic complexity is in the range $O(10.5n^3)$ to $O(16n^3)$. The asymptotic storage requirement is $O(3.5n^2)$. The method also requires square-root evaluations asymptotically in the range of $O(n^2)$ to $O(1.5n^2)$.

Imbedding Algorithms

Jamshidi et al.[36] proposed the imbedded parameter algorithms for the solution of the ARE equation. The three imbedding algorithms they obtained were the Q-imbedding algorithm, S-imbedding algorithm, and the combined imbedding Newton algorithm. Here, we analyze only the Q-imbedding and combined imbedding Newton algorithms.

Q-Imbedding Algorithm

The Q-imbedded Riccati equation is given as

$$\mathcal{H}[P(\epsilon),\epsilon] = P(\epsilon)A(\epsilon) + A^T(\epsilon)P(\epsilon) - P(\epsilon)SP(\epsilon) + \epsilon Q = 0 \tag{39}$$

where

$$S = BR^{-1}B^T \tag{40}$$

and ϵ is the imbedded parameter. Differentiating Eq. (41) with respect to ϵ, we get

$$[A(\epsilon) - SP(\epsilon)]^T \frac{\partial P(\epsilon)}{\partial \epsilon} + \frac{\partial P(\epsilon)}{\partial \epsilon}[A(\epsilon) - SP(\epsilon)] + G(\epsilon) = 0 \tag{41}$$

where $G(\epsilon) = D^T(\epsilon)P(\epsilon) + P(\epsilon)D(\epsilon) + Q$ and $D(\epsilon) = \partial A(\epsilon)/\partial\epsilon$. The parameter ϵ is allowed to vary continuously in [0,1]. Equation (41) is a Lyapunov-type differential equation. Integration of Eq. (41) yields a unique solution of Eq. (39) if certain continuity and differentiability conditions are met. The details are given in Ref. 36. The numerical computation of the imbedded Riccati equation depends on the following three factors:

1) The imbedding of A to make it stable, if it originally is not stable.

2) The numerical solution of the associated Lyapunov equation (41) for $\partial P/\partial\epsilon$.

3) The numerical integration scheme employed to find $P(0 + i\Delta\epsilon)$, $i = 1,\ldots,1/\Delta\epsilon$, where $\Delta\epsilon$ is the choice of step size for ϵ.

Here, we will assume A is originally stable, and therefore, there is no need to imbed A. A is originally stable for structural systems with no rigid body modes. Even otherwise, the step of imbedding is of negligible cost compared to the other steps in the computation. By this assumption, $G(\epsilon) = Q$. The steps in the algorithm are the following:

1) For $\epsilon = 0$, $P = \mathbf{0}$, choose step size $\Delta\epsilon$.

2) Compute necessary matrix $A(\epsilon) - SP(\epsilon)$ from A, S, $P(\epsilon)$.

3) Solve Eq. (41) a sufficient number of times using any of the Lyapunov equation solvers, to perform one integration step from ϵ to $\epsilon + \Delta\epsilon$ to obtain a new approximate $P(\epsilon)$.

4) If $\epsilon < 1$, $\epsilon = \epsilon + \Delta\epsilon$; go to step 2.

5) Stop.

Step 3 of the algorithm can be accomplished by any R-K integration method. For this analysis, we will assume an R-K fourth-order method, which has been used in Ref. 36 for numerical tests, requiring solution of Eq. (41) four times for every step from ϵ to $\epsilon + \Delta\epsilon$. Since the complexity depends mainly on the solution of Eq. (41), the complexity is expressed in terms of $\mathfrak{L}_i(n)$ in line 2 of Table 2. The asymptotic storage for this method os $O(5n^2)$.

Combined Imbedding-Newton Algorithm

Jamshidi et al.[36] after numerous numerical experiments found that a combined imbedding-Newton algorithm is very efficient. The algorithm follows:

1) Choose $\Delta\epsilon = 1$ or 0.5 and obtain an approximate solution $P(1)$ using the Q-imbedding algorithm; set $k = 0$ and $P_k = P(1)$.

2) Compute the Riccati solution using the Newton method given by Eq. (30).

Because both steps 1 and 2 require solving a Lyapunov equation many times, the complexity for this method is given in terms of $\mathfrak{L}_i(n)$ in line 3 of Table 2.

Eigenvector Method

Consider the Hamiltonian matrix

$$Z = \begin{bmatrix} A & -S \\ -Q & -A^T \end{bmatrix} \tag{42}$$

It has been shown[1-3] if Z can be and is made diagonal as $Z = WDW^{-1}$, where

$$D = \begin{bmatrix} \Lambda & \mathbf{0} \\ \mathbf{0} & -\Lambda \end{bmatrix} \tag{43}$$

is a diagonal matrix such that Λ has all eigenvalues on the left half-plane, then the steady-state Riccati matrix is given by $P = W_{21}W_{11}^{-1}$, where W_{21} and W_{11} are the partitions of W as shown next.

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \tag{44}$$

If the eigenvalues are complex-conjugate pairs, the transformation W is complex and the computation becomes expensive. Fath[28] extended the method so that the transformation W remained real by obtaining D in a block-diagonal form. In this work, Fath's extended eigenvector algorithm is analyzed for its complexity. The steps in the algorithm are the following:

1) Reduce the Hamiltonian matrix if it is reducible.

2) Scale the reduced matrix using the procedure for scaling given in Ref. 28 and accumulate the scaling transformations. This scaling is done to minimize the Euclid norm of the matrix so as to reduce errors in later operations.

3) Transform the scaled matrix to upper Hessenberg form.

4) Compute eigenvalues using a double QR algorithm.

5) Compute the eigenvectors of left half-plane eigenvalues, using inverse iteration with shifts on the Hessenberg matrix equal to the computed left half-plane eigenvalues.

6) Transform the eigenvectors of the Hessenberg matrix to those of the Hamiltonian matrix.

7) Compute Riccati matrix $P = W_{21}W_{11}^{-1}$.

For our analysis, we assume that the Hamiltonian matrix is irreducible. Even otherwise, the reducing step just involves shuffling rows and columns, but does not involve any computation. The steps 4 and 5 have varying complexities depending on the spectrum of Z. We will consider the two extreme cases of all real and all complex-conjugate pairs of eigenvalues. The total complexity and storage requirement are given in lines 4(a) and (b) of Table 2. For details on how this complexity is arrived at, see Ref. 50. The asymptotic storage requirement for this method is $O(14n^2)$.

Schur Vector Method

Laub[24] proposed the solution to the ARE in terms of the Schur vectors of the Hamiltonian matrix. Holley and Wei[25] also proposed a solution to the ARE along similar lines.

If the Hamiltonian matrix Z as in Eq. (42) is transformed using orthogonal similarity transformations to the upper real Schur form

$$Z' = U^TZU = \begin{bmatrix} Z'_{11} & Z'_{12} \\ \mathbf{0} & Z'_{22} \end{bmatrix}$$

where Z' is quasi-upper-triangular with possible 2×2 blocks on the diagonal, and if Z'_{11} has all eigenvalues in the left half-plane, then if U is partitioned as

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}$$

the solution to the algebraic matrix Riccati equation is given as $P = U_{21}U_{11}^{-1}$. For proof, see Ref. 24.

It is well known that the greatest advantage in the Schur method is the numerical stability of the Schur transformations using the double QR algorithm. The steps in the algorithm are the following:

1) Transform Z to upper Hessenberg form.

2) Accumulate the transformations of step 1.

3) Transform the upper Hessenberg matrix to upper real Schur form using a double QR algorithm.

4) Accumulate the Schur transformations with the previous transformation.

5) Reorder, if necessary, the upper real Schur form to get the block Z'_{11} with all left half-plane eigenvalues.

6) Accumulate the reordering transformations with previous transformations to obtain U.

7) Solve for the Riccati matrix from the equation $U_{11}^TP = U_{21}^T$.

Steps 3-6 have varying complexities depending on the spectrum of Z. We will consider the two extreme cases, case (a) of all complex-conjugate eigenvalue pairs and case (b) of all real eigenvalues. The total complexity and storage requirement are given in lines 5(a) and (b) of Table 2. For more details on how these expressions are arrived at, see Ref. 50. The asymptotic storage estimate for this method is $O(8n^2)$.

Matrix Sign-Function Solution

Roberts[20] proposed the Riccati solution to the ARE in terms of the matrix sign function of the Hamiltonian matrix Z in Eq. (42). The sign function of any square matrix A that has a Jordan form $A = U(D + N)U^{-1}$, where U is the matrix of generalized eigenvectors, D the diagonal matrix of eigenvalues, and N a nilpotent matrix that commutes with D, is defined as

$$\text{sign}(A) = U\,\text{diag}[\text{sgn}(d_1),\ldots,\text{sgn}(d_n)]U^{-1}$$

where sgn is the scalar sign function

$$\text{sgn}(z) = \begin{cases} 1; & \text{if } Re(z)>0 \\ -1; & \text{if } Re(z)<0 \end{cases}$$

The sign function has the following important properties that can be easily verified:

1) $\text{sign}(A)^2 = I$ where I is the identity matrix.

2) $\lambda[\text{sign}(A)] \in \{1, -1\}$ where $\lambda[\text{sgn}(A)]$ is any eigenvalue of $\text{sgn}(A)$.

3) $\text{sign}(cA) = c\ \text{sign}(A)$.

4) $\text{sign}(TAT^{-1}) = TAT^{-1}$.

Using property 4 and the stabilizability of the state matrix A, Roberts gave the solution P to the ARE as

$$\begin{bmatrix} W_{12} \\ W_{22}+I \end{bmatrix} P = -\begin{bmatrix} W_{11}+I \\ W_{21} \end{bmatrix} \tag{45}$$

where

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \text{sign}(Z)$$

For proof, see Ref. 20. Equation (45) can be least-squares solved for P. Therefore, $\text{sign}(Z)$ has to be found efficiently to obtain the Riccati matrix. Bierman[21] gives the following efficient algorithm to obtain $\text{sign}(Z)$:

$$Z_0 = JZ$$

$$Z_{k+1} = \frac{1}{2c}[Z_k + c^2 JZ_k^{-1}J]; \quad c = |\det Z_k|^{1/2N}; \quad k = 0,\ldots,k_s$$

$$Z_{k_s} \to J\ \text{sign}(Z)$$

where

$$J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}$$

The least-squares solution of Eq. (45) may differ from the actual solution because of roundoff and Z_{k_s} not being exactly equal to $\text{sign}(Z)$. If $P = P_{ls} + \Delta P$, where P_{ls} is the computed least-squares solution of Eq. (45), the correction ΔP can be shown to satisfy the following algebraic Riccati equation:

$$(A - SP_{ls})^T \Delta P + \Delta P(A - SP_{ls}) - \Delta P S \Delta P + \mathcal{F}(P_{ls}) = 0 \tag{46}$$

$\mathcal{F}(\cdot)$ is the function in Eq. (4), which is the residual matrix of the algebraic Riccati equation. Bierman[21] suggests solving Eq. (46) for ΔP using the sign-function algorithm. This type of correction can be done many times, each time solving an algebraic Riccati equation using the sign-function algorithm until an accurate solution P is obtained. Let k_s be the number of iterations of the sign-function algorithm and k_d be the number of corrections. Then, the computational complexity of the sign-function solution to the ARE is asymptotically equal to $O[(20k_s + 19)(k_d + 1)(n^3/3)]$. The detailed complexity and storage requirement are given in line 6 of Table 2.

Comparison of the Algorithms

RDE Algorithms

Since all algorithms for the RDE involve some integration step size parameter that may vary widely depending on the problem and the accuracy of solution desired, comparison between the speed of these algorithms can be reliably made only by means of extensive numerical testing. However, the complexity expressions derived here give the minimum sampling time for any algorithm. For example, the eigenfactor algorithm requires a minimum time of $O(n^4)$, which may not be suitable for some fast control applications. The storage expressions also show the limits on the order of systems that can be controlled using these algorithms.

ARE Algorithms

For the ARE algorithms, reliable comparison can be made and conclusions drawn on the speed of the algorithms, based on the complexity expressions.

First, we compare the Lyapunov equation solvers to pick the best one for the Newton method. If we use the typical value of $k_1 = 10$ to 12 for methods 1 and 2, which has been suggested as adequate by the authors of method 2, and using $k_2 = 5$ to 6 for method 3 as suggested by the authors of method 3, and a value of $q = 2$ for methods 4 and 5, which Wilkinson[51] suggests as a reasonable estimate for most matrices in their double QR reduction, the following approximate, but reasonable estimates can be made.

Method	Asymptotic complexity	Asymptotic storage
1	$O(27n^3)$-$O(33n^3)$	$O(3.5n^2)$
2	$O(26n^3)$-$O(32n^3)$	$O(3.5n^2)$
3	$O(16n^3)$-$O(17n^3)$	$O(3.5n^2)$
4	$O(11n^3)$-$O(16n^3)$	$O(2.5n^2)$
5	$O(10.5n^3)$-$O(16n^3)$	$O(3.5n^2)$

The methods 4 and 5 require, in addition, $O(0.5n^2)$ to $O(n^2)$ and $O(n^2)$ to $O(1.5n^2)$ square-root evaluations, respectively. From the foregoing expressions, we can conclude that either method 4 or 5 is most suitable for use as a subroutine for the Newton method and the imbedded solutions method. Method 5 is to be preferred as it guarantees a positive semidefinite solution with little or no extra cost over method 4. Numerical tests[49] have shown method 4 to be superior in speed and accuracy over methods 1 and 2.

The Schur method has an asymptotic complexity anywhere in the range of $O[n^3(20 + 53q/2)]$ to $O[n^3(20 + 89q/2)]$, which becomes with $q = 2$ equal to $O(73n^3)$ to $O(109n^3)$. Out of this, the QR reduction step and accumulation of Schur transformations take about 25-35% each. The reordering of the eigenvalues takes anywhere between 0-20%. Without any reordering, the complexity range drops to $O(60n^3)$ to $O(100n^3)$. The eigenvector method, on the other hand, has a complexity asymptotically in the range $O[(68/3 + 10q + 4q')n^3]$ to $O[(62/3 + 20q + 2q')n^3]$. If we use $q = 2$ and $q' = 2$, which is typical for inverse iteration with previously determined eigenvalues,[51] the asymptotic estimate becomes $O(50n^3)$ to $O(65n^3)$. This shows that, in general, the eigenvector method will be faster than the Schur method. This is so, even after assuming no necessity for reordering in the Schur method, which usually is not true. On the aspect of storage, the eigenvector method does miserably compared to the Schur vector method, requiring $O(14n^2)$ compared to $O(8n^2)$, a 75% larger storage requirement over the Schur method.

If we compare the Newton algorithm using either Lyapunov solver 4 or 5, with the Schur vector method, it can be concluded that the Newton method will be faster, only if it requires a $k_{\max} \leq 8$. Since for large matrices, this number is quite small, particularly if the initial guess P_0 is bad, the Schur vector method will be faster than the Newton method for large systems. Thus, the eigenvector method will also be faster than the Newton method. The Q-imbedding algorithm using the

R-K fourth-order integration requires $O(40n^3/\Delta\epsilon)$ to $O(65n^3/\Delta\epsilon)$, that is, about $4/\Delta\epsilon$ times the complexity in the Newton method. Since $\Delta\epsilon$ has to be quite small, the Schur method and the eigenvector method will be, in general, much faster than the Q-imbedding algorithm. The combined imbedding-Newton algorithm requires $O[10(8 + k'_{\max})n^3]$ to $O[16(8 + k'_{\max})n^3]$, where $k'_{\max}$ is the number of Newton iterations. Even with $k'_{\max} = 1$, the combined imbedding-Newton method does worse than the eigenvector and Schur methods.

The matrix sign-function algorithm with one correction pass, that is, $k_d = 1$ and one sign-function iteration, $k_s = 1$, has complexity $O(26n^3)$. However, one sign-function iteration will rarely be sufficient to get the sign-function iterate to be close to the exact sign function and k_s may even be of the order of 20,[21] and $k_d = 1$ also may not be sufficient for an accurate solution. Therefore, the matrix sign-function algorithm will, in general, be slower than the eigenvector and Schur methods. This method requires storage $O(10.5n^2)$ that is more than that required for the Schur method but less compared to the requirement of the eigenvector method.

If we compare the integration methods with the Schur and eigenvector methods, it can be concluded that the direct integration using the kth-order R-K method will be faster only if it requires a time step such that s is less than about $(80/\beta)$ for the Schur method and less than about $(60/\beta)$ for the eigenvector method. Since usually the direct integration is numerically unstable and requires a very small step size, these numbers are quite small, if steady-state solution is to be reached by integration. Therefore for large systems, the Schur vector method and the eigenvector method will be much faster than direct integration to steady state.

Summary

The computational complexities and storage requirements are given for a few selected Riccati equation solvers. These expressions do not reflect the numerical stability of the algorithms and accuracy of the solutions they yield. These two aspects together with the complexity and storage determine the best algorithm. Comparison of these complexity expressions is then done, whenever such a comparison leads to reliable conclusions as in the case of the algebraic Riccati equation algorithms. For the Riccati differential equation (RDE) algorithms, the complexity expressions cannot be so easily compared to identify the faster algorithms because of the integration step size parameters in these expressions that may vary widely depending on the problem and accuracy desired. However, these complexity expressions give the minimum sampling time for each of the RDE algorithms. The complexity expressions given here are also necessary to compute the speedup and efficiency of any implementation of these algorithms on concurrent machines.

For large-order systems as in large space structure control, concurrent processing will become absolutely essential because of storage limitations and the need to have lesser sampling times. The complexity study done here has also given us more insight into the computational flow of these algorithms, which helps in identifying the degree of concurrency present in them.

Acknowledgments

The research herein was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA and was presented at the ARO/AFOSR Conference on Non-Linear Vibrations, Stability, and Dynamics of Structures and Mechanisms, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, March 23-25, 1987.

Appendix: Guide to Tables 1, 2, and 3

k_d = number of least-squares corrections or number of passes in the matrix sign-function algorithm

k_s = average number of iterations per pass to obtain the sign function of the Hamiltonian matrix in the matrix sign-function algorithm

$k_{\max}$ = number of Newton iterations in Newton's method for the ARE

$k'_{\max}$ = number of Newton iterations in combined imbedding-Newton algorithm

k_1 = number of iterations for convergence of series solution in Lyapunov equation solvers 4 and 5

k_2 = number of iterations in Lyapunov equation solver 6

m = number of control variables

n = number of state variables

q = average number of double QR iterations required to make a subdiagonal element zero

q' = average number of iterations required to obtain any eigenvector from its known eigenvalue

p = number of sensors

s = number of time stations

s' = number of time stations in Davison-Maki algorithm

s'' = number of scaling transformations in Fath's algorithm

$\mathfrak{L}_i(n)$ = computational complexity of Lyapunov solver i

$\mathfrak{M}_i(n)$ = storage requirement of Lyapunov solver i

α = number that depends on the particular ***R-K*** integrator being used

α' = number that depends on the particular ***R-K*** integrator being used

β = number of function evaluations required by the particular ***R-K*** method being used

$\Delta\epsilon$ = step size of imbedding parameter in the Q-imbedding method

Note: All the algorithms marked by an asterisk are assumed to use some ***R-K*** integrator that requires β function evaluations for the differential equations involved. The storage expressions given for these algorithms correspond to the fourth-order ***R-K*** method, that is, $\beta = 4$. For the fourth-order method $\alpha = 4$, $\alpha' = 3$.

References

[1]MacFarlane, A. G. J., "An Eigenvector Solution to Optimal Linear Regulator Problem," *Journal of Electronics and Control*, Vol. 14, 1963, pp. 643-654.

[2]Potter, J. E., "Matrix Quadratic Solutions," *SIAM Journal of Applied Mathematics*, Vol. 14, 1966, pp. 496-501.

[3]O'Donnell, J. J., "Asymptotic Solution of Matrix Riccati Equation of Optimal Control," *Proceedings of the 4th Allerton Conference on Circuits and Systems*, Dept. of Electrical Engineering, Univ. of Illinois, Urbana, IL, 1966, pp. 577-586.

[4]Kalman, R. E. and Englar, T. S., "A Users' Manual for the Automatic Synthesis Program," NASA CR-475, June 1966.

[5]Blackburn, T. R., "Solution of the Algebraic Matrix Riccati Equation via Newton-Raphson Iteration," *AIAA Journal*, Vol. 6, May 1968, pp. 951-953.

[6]Kleinman, D. L, "On the Iterative Technique for Riccati Equation Computations," *IEEE Transactions on Automatic Control*, Vol. AC-13, 1968, pp. 114-115.

[7]Bryson, A. E. and Hall, W. E. Jr., "Optimal Control and Filter Synthesis by Eigenvector Decomposition," Center for Systems Research, Stanford Univ., Stanford, CA, TR-436, Nov. 1971.

[8]Vaughan, D. R., "A Negative Exponential Solution for the Matrix Riccati Equation," *IEEE Transactions on Automatic Control*, Vol. AC-14, Feb. 1969.

[9]Lainiotis, D. G., "New Results on the Solution of Matrix Riccati Equations," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1973.

[10]Lainiotis, D. G., "Partitioned Riccati Algorithms," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1975.

[11]Lainiotis, D. G., "Partitioned Riccati Solutions and Integration Free Doubling Algorithms," *IEEE Transactions on Automatic Control*, Vol. AC-21, Oct. 1976, pp. 677-688.

[12]Lainiotis, D. G., "Generalized Chandrasekhar Algorithms: Time Varying Models," *IEEE Transactions on Automatic Control*, Vol. AC-21, Oct. 1976, pp. 728-732.

[13]Womble, M. E. and Potter, J. E., "A Prefiltering Version of the Kalman Filter with New Integration Formulas for Riccati Equations," *IEEE Transactions on Automatic Control*, Vol. AC-20, June 1975, pp. 378-380.

[14]Casti, J., "Matrix Riccati Equations, Dimensionality Reduction, and Generalized *X-Y* Functions," *Utilitas Mathematica*, Vol. 6, 1974, pp. 95-110.

[15]Kailath, T., "Some Chandrasekhar Type Algorithms for Quadratic Regulation," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1972.

[16]Kailath, T., "Some New Algorithms for Recursive Estimation in Constant Linear Systems," *IEEE Transactions on Information Theory*, Vol. IT-19, Nov. 1973, pp. 750-760.

[17]Morf, M., Levy, B., and Kailath, T., "Square-Root Algorithms for the Continuous Time Least-Square Estimation Problem," *IEEE Transactions on Automatic Control*, Vol. AC-28, Oct. 1978, pp. 907-911.

[18]Brammer, R. F., "A Note on Use of Chandrasekhar Equations for the Calculation of Kalman Gain Matrix," *IEEE Transactions on Information Theory*, Vol. IT-21, May 1975, pp. 334-337.

[19]Martenson, K., "On the Matrix Riccati Equation," *Information Sciences*, Vol. 3, 1971, pp. 17-49.

[20]Roberts, J. D., "Linear Model Reduction and Solution of Algebraic Riccati Equation by Use of the Sign Function," *International Journal of Control*, Vol. 32, 1980, pp. 677-687.

[21]Bierman, G. J., "Computational Aspects of the Matrix Sign Function Solution to the ARE," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1984.

[22]Gardiner, J. D. and Laub, A. J., "A Generalization of the Matrix Sign Function Solution for Algebraic Riccati Equations," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1985.

[23]Arnold, W. F. and Laub, A. J., "Generalized Eigenproblem Algorithm and Software for Algebraic Riccati Equations," *Proceedings of the IEEE*, Vol. 72, Institute of Electrical and Electronic Engineers, New York, Dec. 1984, pp. 1746-1754.

[24]Laub, A. J., "A Schur Method for Solving Algebraic Riccati Equations," *IEEE Transactions on Automatic Control*, Vol. AC-24, 1979, pp. 913-921.

[25]Holley, W. E. and Wei, S. Y., "An Improvement in the MacFarlane-Potter Method for Solving the Algebraic Riccati Equation," *Proceedings of the JACC*, American Institute of Chemical Engineering, New York, June 1979, pp. 921-923.

[26]Yaakov, O. and Bar-Itzhack, I. Y., "Eigenfactor Solution of Matrix Riccati Equation - A Continuous Square Root Algorithm," *IEEE Transactions on Automatic Control*, Vol. AC-30, Oct. 1985, pp. 1104-1113.

[27]Bunse-Gesstner, A. and Mehrmann, V., "A Symplectic QR Like Algorithm for the Solution of the Real Algebraic Riccati Equation," *IEEE Transactions on Automatic Control*, Vol. AC-31; Dec. 1986.

[28]Fath, A. F., "Computational Aspects of Linear Optimal Regulator Problem", *IEEE Transactions on Automatic Control*, Vol. AC-14, 1969, pp. 547-550.

[29]Greenberg, S. G. and Bard, Y., "A Comparison of Computational Methods for Solving the Algebraic Matrix Riccati Equation," *Proceedings of the IEEE Decision and Control Conference*, Institute of Electrical and Electronic Engineers, New York, Dec. 1971.

[30]Farrar, F. A. and DiPietro, R. C., "Comparative Evaluation of Numerical Methods for Solving the Algebraic Matrix Riccati Equation," United Technologies Research Center, East Hartford, CT, Rept. R76-140268-1, Dec. 1976.

[31]Davison, E. J. and Maki, M. C., "The Numerical Solution of the Matrix Riccati Differential Equation," *IEEE Transactions on Automatic Control*, Vol. AC-18, Feb. 1973.

[32]Taylor, F., "Comments on the Numerical Solution of the Matrix Riccati Differential Equation," *IEEE Transactions on Automatic Control*, Vol. AC-19, Feb. 1974.

[33]Casti, J. and Kirscner, O., "Numerical Experiments in Linear Control Theory Using Generalized *X-Y* Equations," *IEEE Transactions on Automatic Control*, Vol. AC-21, Oct. 1976.

[34]Kenney, C. S. and Leipnik, R. B., "Numerical Integration of the Differential Matrix Riccati Equation," *IEEE Transactions on Automatic Control*, Vol. AC-30, Oct. 1985.

[35]Arnold, W. F., "Numerical Solution of Algebraic Matrix Riccati Equations," Naval Weapons Center Rept., China Lake, CA, Feb. 1984.

[36]Jamshidi, M. and Bottiger, F. "On the Imbedded Solutions of Algebraic Riccati Equation," *Proceedings of the JACC*, American Society of Mechanical Engineers, New York, 1976, pp. 474-481.

[37]Tapley, B. D. and Choe, C. Y., "An Algorithm for Propagating the Square-Root Covariance Matrix in Triangular Form", *IEEE Transactions on Automatic Control*, Vol. AC-21, Feb. 1976, pp. 122-123.

[38]Andrews, A., "A Square Root Formulation of the Kalman Covariance Equations," *AIAA Journal*, Vol. 6, June 1968, pp. 1165-1166.

[39]Neudecker, H., "A Note on Kronecker Matrix Products and Matrix Equation Systems," *SIAM Journal of Applied Mathematics*, Vol. 17, May 1969, pp. 603-606.

[40]Chen, C. F. and Shieh, L. S., "A Note on Expanding $PA + A'P = -Q$," *IEEE Transactions on Automatic Control*, Vol. AC-13, Feb. 1968, pp. 122-123.

[41]Bingulac, S. P., "An Alternate Approach to Expanding $PA + A'P = -Q$," *IEEE Transactions on Automatic Control*, Vol. AC-15, Feb. 1970, pp. 135-137.

[42]Lu, C. S., "Solution of the Matrix Equation $AX + XB = C$," *Electronics Letters*, April 1971.

[43]Jameson, A., "Solution of the Equation $AX + XB = C$ by Inversion of an $M \times M$ or $N \times N$ Matrix," *SIAM Journal of Applied Mathematics*, Sept. 1968.

[44]Smith, R. A., "Matrix Equation $XA + BX = C$," *SIAM Journal of Applied Mathematics*, Vol. 16, 1968, pp. 199-201.

[45]Davison, E. J. and Man, F. T., "The Numerical Solution of $A'Q + QA = -C$," *IEEE Transactions on Automatic Control*, Vol. AC-13, Aug. 1968, pp. 448-449.

[46]Hoskins, W. D., Mark, D. S., and Walton, D. J., "The Numerical Solution of $A'Q + QA = -C$," *IEEE Transactions on Automatic Control*, Vol. AC-22, Oct. 1977.

[47]Bartels, R. H. and Stewart, G. W., "Solution of Matrix Equation $AX + XB = C$," *Communications of the ACM*, Sept. 1972.

[48]Hammarling, S. J., "Numerical Solution of the Stable, Nonnegative Definite Lyapunov Equation," *IMA Journal of Numerical Analysis*, Vol. 2, 1982, pp. 303-323.

[49]Belanger, P. R., and McGillivray, T. D., "Computational Experience with the Solution of the Matrix Lyapunov Equation," *IEEE Transactions on Automatic Control*, Vol. AC-21, Oct. 1976.

[50]Ramesh, A. V., "Computational Complexities in Optimal Control," M.S. Thesis, Dept. of Civil and Environmental Engineering, Duke Univ., Durham, NC, 1987.

[51]Wilkinson, J. H., *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London, 1965.

[52]Kwakernaak, H. and Sivan, R., *Linear Optimal Control Systems*, Wiley, New York, 1972.

[53]Golub, G. H. and Van Loan, C. F., *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, MD, 1983.

[54]Moler, C. B. and Van Loan, C. F., "Nineteen Dubious Ways to Compute the Exponential of a Matrix," *SIAM Review*, Vol. 20, 1978, pp. 801-836.