Parallel Solution of Large-Scale and Sparse Generalized algebraic Riccati Equations

José M. Badía[1], Peter Benner[2], Rafael Mayo[1], and Enrique S. Quintana-Ortí[1]

[1] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain; {badia,mayo,quintana}@icc.uji.es.
[2] Fakultät für Mathematik, Technische Universität Chemnitz, D-09107 Chemnitz, Germany; benner@mathematik.tu-chemnitz.de.

Abstract. We discuss a parallel algorithm for the solution of large-scale generalized algebraic Riccati equations with dimension up to $\mathcal{O}(10^5)$. We survey the numerical algorithms underlying the implementation of the method, in particular, a Newton-type iterative solver for the generalized Riccati equation and an LR-ADI solver for the generalized Lyapunov equation. Experimental results on a cluster of Intel Xeon processors illustrate the benefits of our approach.

Key words: generalized algebraic Riccati equation, Newton's method, generalized Lyapunov equation, LR-ADI iteration, parallel algorithms.

1 Introduction

Consider the (generalized) *algebraic Riccati equation* (ARE)

$$0 = A^T XE + E^T XA - E^T XBR^{-1}B^T XE + C^T QC =: \mathcal{R}(X), \qquad (1)$$

where $A,\ E \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $R \in \mathbb{R}^{m\times m}$ is symmetric positive definite, and $Q \in \mathbb{R}^{p\times p}$ is symmetric positive semidefinite. Under certain conditions, the ARE (1) has a unique symmetric positive semidefinite solution $X \in \mathbb{R}^{n\times n}$ [17, 19]. This particular solution is usually required in applications.

Solving the ARE (1) is the key step in many computational methods for model reduction, filtering, and controller design of dynamical linear systems (see, among many, [10, 13–15, 19, 20, 24] and the references therein). In general, numerical methods for solving AREs have a computational cost of $\mathcal{O}(n^3)$ floating-point arithmetic operations (flops) and require storage for $\mathcal{O}(n^2)$ numbers [17, 19, 24]. While current desktop computers provide enough computational power to solve problems with state-space dimension n in the hundreds using libraries such as SLICOT (`http://www.slicot.org`) or the Matlab control-related toolboxes, large-scale applications clearly require the use of advanced computing techniques.

Over the last few years we have developed a library of parallel algorithms for the solution of (dense) AREs on parallel architectures [6]. The library, PLiCOC, employs the kernels in LAPACK, BLAS, and ScaLAPACK [3, 8], enabling the solution of equations with n in the order of a few thousands. However, this approach is still insufficient for very large-scale AREs arising in weather forecast, circuit simulation, VLSI chip design, and air quality simulation, among others (see, e.g., [4, 9, 11, 12]). Dynamical systems leading to AREs with dimension n as high as $\mathcal{O}(10^4) - \mathcal{O}(10^5)$ and sparse matrix pairs (A, E) are common in these applications.

In this paper we consider a different method that exploits the sparse structure of the matrix pair (A, E) in (1), and thus allows the solution of much larger AREs. (Throughout the paper the term "sparse" will refer to both unstructured sparse matrices and structured ones, such as banded matrices.) The codes employ the parallel kernels in ScaLAPACK [8]. Depending on the specific structure of A and E, the (unstructured) sparse linear system solvers in MUMPS [2] or the banded linear system solver in ScaLAPACK are also employed.

The rest of the paper is structured as follows. In Section 2 we review a specialized formulation of Newton's iterative method for the solution of large-scale sparse AREs. The major computational task in this method is the solution of a large-scale sparse (generalized) Lyapunov equation. A variant of the iterative Lyapunov solver introduced in [18, 22], based on a low-rank *alternating direction implicit* (LR-ADI) method, is then summarized in Section 3. Following the description of the numerical methods, in Section 4 we offer some details on the parallelization of the corresponding algorithms. In Section 5, experiments on a cluster of Intel Xeon processors report the potential of the ARE solver. Finally, we give some concluding remarks in Section 6.

2 Newton's method for the ARE

In this section we review a variant of Newton's method, described in [23], which delivers a full-rank approximation of the solution of large-scale sparse AREs. Here we focus on the implementation details relevant to an efficient (parallel) implementation.

Starting from an initial solution X_0, Newton's method for the ARE [16] proceeds as follows:

```
Newton's method
1) Compute the Cholesky factorization $Q = \bar{Q}\bar{Q}^T$
2) Compute the Cholesky factorization $R = \bar{R}\bar{R}^T$
3) $\bar{C} := \bar{Q}^T C$
4) $\bar{B} := E^{-1}BR^{-1} = ((E^{-1}B)\bar{R}^{-T})\bar{R}^{-1}$
repeat with $j := 0, 1, 2, \ldots$
    5) $K_j := E^T X_j B R^{-1} = E^T X_j E \bar{B}$
    6) $\hat{C}_j := \begin{bmatrix} \bar{C} \\ \bar{R}^T K_j^T \end{bmatrix}$
    7) Solve for $X_{j+1}$:
        $0 = (A - BK_j^T)^T X_{j+1} E + E^T X_{j+1} (A - BK_j^T) + \hat{C}_j^T \hat{C}_j$
until $\|X_j - X_{j-1}\| < \tau \|X_j\|$
```

Provided $(A - BR^{-1}B^T X_0, E)$ is a stable matrix pair (i.e., all its eigenvalues lie on the open left half plane), this iteration converges quadratically to the desired symmetric positive semidefinite solution of the ARE [16], $X_\infty = \lim_{j\to\infty} X_j$. In practice, (A, E) is often a stable matrix pair, so that setting $X_0 := 0$ is enough to guarantee the convergence of the iteration. Thus, no globalization strategy is required to guarantee convergence. Note that a line search procedure in [7] can be used to accelerate initial convergence, though.

In real large-scale applications, $m, p \ll n$, and both A and E are sparse, but the solution matrix X is in general dense and, therefore, impossible to construct explicitly. However, X is often of low-numerical rank and thus can be approximated by a full-rank factor $\hat{R} \in \mathbb{R}^{n\times r}$, with $r \ll n$, such that $\hat{R}\hat{R}^T \approx X$. The method described next aims at computing this "narrow" factor $\hat{R}$ instead of the explicit solution.

2.1 Exploiting the rank-deficiency of the solution

Let us review how to modify Newton's method in order to avoid explicit references to X_j. Note that all but one of the computations in Steps 1–4 of Newton's method involve matrices of small dimensions and therefore can be performed employing dense linear algebra kernels even if the matrices are sparse. In particular, the cost of the two Cholesky factorizations in Steps 1 and 2 is $m^3/3 + p^3/3$ flops, and obtaining $\bar{C} \in \mathbb{R}^{p\times n}$ from there in Step 3 requires n^2p additional flops. On the other hand, computing $\bar{B} \in \mathbb{R}^{n\times m}$ in Step 4 requires $2m^2n$ flops plus the cost of solving the system $E^{-1}B$. The cost of this latter operation (via, e.g., a direct method) strongly depends on the sparsity degree and pattern of the coefficient matrix E, and the solver that is employed. For unstructured sparse matrices, this cost is difficult to determine *a priori.*

Assume for the moment that, at the beginning of the iteration, we maintain $\hat{R}_j \in \mathbb{R}^{n\times r_j}$ such that $E^{-T}\hat{R}_j\hat{R}_j^T E^{-1} = X_j$. Then, in the first step of the iteration, we can compute K_j as

$$K_j := E^T X_j E\bar{B} = \hat{R}_j(\hat{R}_j^T\bar{B}),$$

which initially requires a (dense) matrix product, $M := \hat{R}_j^T\bar{B}$, at a cost of $2r_jmn$ flops, and then a (dense) matrix product, $\hat{R}_jM$, with the same cost. Even for large-scale problems, as m is usually a small order constant, this represents at most a quadratic cost. In practice, r_j usually remains a small value during the iteration so that this cost becomes as low as linear.

The matrix product $\bar{R}^T K_j^T$ needed in the second step of the iteration for the construction of $\hat{C}_j$ presents only a moderate cost, $2m^2n$ flops, and therefore does not require any special action.

The key of this approach lies in solving the Lyapunov equation in the third step for a full-rank factor $\hat{R}_{j+1}$, such that $E^{-T}\hat{R}_{j+1}\hat{R}_{j+1}^T E^{-1} = X_{j+1}$. We describe how to do so in the next section.

3 Low Rank Solution of Lyapunov Equations

In this section we introduce a generalization of the Lyapunov solver proposed in [18, 22], based on the cyclic *low-rank alternating direction implicit* (LR-ADI) iteration.

Consider the Lyapunov equation to be solved at each iteration of Newton's method

$$0 = (A - BK^T)^T Y E + E^T Y (A - BK^T) + \hat{C}^T \hat{C}, \qquad (2)$$

where, for simplicity, we drop all subindices in the expression. Here, $A,\ E \in \mathbb{R}^{n\times n}$, $B, K \in \mathbb{R}^{n\times m}$, and $\hat{C} \in \mathbb{R}^{(p+m)\times n}$. Recall that we are interested in finding a full-rank factor $S \in \mathbb{R}^{n\times s}$, with $s \ll n$, such that $SS^T \approx Y$. Then, in the jth iteration of Newton's method, $\hat{R}_j := S$ and $r_j := s$.

A generalization of the LR-ADI iteration, tailored for equation (2), can be formulated as follows:

```
LR-ADI iteration
1) $V_0 := ((A - BK^T)^T + \sigma_1 E^T)^{-1} \hat{C}^T$
2) $S_0 := \sqrt{-2\ \alpha_1}\ V_0$
repeat with $l := 0, 1, 2, \ldots$
    3) $V_{l+1} := V_l - \delta_l ((A - BK^T)^T + \sigma_{l+1} E^T)^{-1} V_l$
    4) $S_{l+1} := [S_l\ ,\ \gamma_l V_{l+1}]$
until $\|\gamma_l V_l\|_1 < \tau \|S_l\|_1$
```

In the iteration, $\{\sigma_1, \sigma_2, \ldots\}$, $\sigma_l = \alpha_l + \beta_l\,\jmath$, is a cyclic set of (possibly complex) shift parameters (that is, $\sigma_l = \sigma_{l+t}$ for a given period t), $\gamma_l = \sqrt{\alpha_{l+1}/\alpha_l}$, and $\delta_l = \sigma_{l+1} + \overline{\sigma_l}$, with $\overline{\sigma_l}$ the conjugate of σ_l. The convergence rate of the LR-ADI iteration strongly depends on the selection of the shift parameters and is super-linear at best [18, 22, 26].

At each iteration the column dimension of S_{l+1} is increased by $(p + m)$ columns with respect to that of S_l so that, after $\bar{l}$ iterations, $S_{\bar{l}} \in \mathbb{R}^{n\times \bar{l}(p+m)}$. For details on a practical criterion to stop the iteration, see [5, 22]. Note that the LR-ADI iteration does not guarantee full colum rank of the S_l. This could be achieved using a column compression based on a rank-revealing LQ factorization. As the full-rank property is irrelevant for the approximation quality, we will not discuss this any further. Possible positive effects on the efficiency of the algorithm will be reported elsewhere.

From the computational view point, the iteration only requires the solution of linear systems of the form

$$((A - BK^T)^T + \sigma E^T)V = W \quad \Leftrightarrow \quad ((A + \sigma E) - BK^T)^T V = W, \qquad (3)$$

for V. Now, even if A and E are sparse (and therefore, so is $\bar{A} := A + \sigma E$), the coefficient matrix of this linear system is not necessarily sparse. Nevertheless, we can still exploit the sparsity of $A,\ E$ by relying on the *Sherman-Morrison-Woodbury (SMW) formula*

$$(\bar{A} - BK^T)^{-1} = \bar{A}^{-1} + \bar{A}^{-1} B (I_m - K^T \bar{A}^{-1} B)^{-1} K^T \bar{A}^{-1}.$$

Specifically, the solution V of (3) can be obtained following the next five steps:

```
SMW formula
1) $V := \bar{A}^{-T}W$
2) $T := \bar{A}^{-T}K$
3) $F := I_m - B^T T$
4) $T := TF^{-1}$
5) $V := V + T(B^T V)$
```

Steps 1 and 2 require the solution of two linear systems with sparse coefficient matrix $\bar{A}$. The use of direct solvers is recommended here as iterations l and $l+t$ of the LR-ADI method share the same coefficient matrices for the linear system. The remaining three steps operate with dense matrices of small-order; specifically, $F \in \mathbb{R}^{m\times m}$, $T \in \mathbb{R}^{n\times m}$ so that Steps 3, 4, and 5 only require $2m^2n$, $2m^3/3 + m^2n$, and $4mn(m+p)$ flops, respectively.

4 Parallel Implementation

The numerical algorithms described in the previous two sections for Newton's method and the LR-ADI iteration are composed of a few dense linear algebra operations (Cholesky factorizations, matrix products, and linear systems) involving dense matrices of relatively small order, and the solution of sparse linear systems (via direct methods) with large-scale coefficient matrices.

Our approach for dealing with these matrix operations is based on the use of existing parallel linear algebra and communication libraries. In Fig. 1 we illustrate the multilayered architecture of libraries employed by our ARE solver included in *SpaRed* (a parallel library for model reduction of large-scale sparse linear systems; `http://www.pscom.uji.es/modred/SpaRedW3/SpaRed.html`).

The solver employs the parallel kernels in ScaLAPACK. Depending on the structure of the state matrix pair (A, E) the banded linear system solver in ScaLAPACK or the sparse linear system solvers in MUMPS are also invoked. Table 1 lists the specific routines employed for each one of the major operations in the algorithm. In the table we do not include the operations required to compute the shift parameters for the LR-ADI iteration as that part of the algorithm was not described. The shifts are currently obtained using an Arnoldi-type iteration. The implementation employs PARPACK, and requires the solution of sparse linear systems and the sparse matrix-vector product. For the first operation, depending on the structure of the coefficient matrix we again use ScaLAPACK or MUMPS. The sparse matrix-vector product is parallelized as a sequence of dot products with the matrix cyclically distributed by rows.

5 Experimental Results

All the experiments presented in this section were performed on a cluster of $n_p = 16$ nodes using IEEE double-precision floating-point arithmetic ($\varepsilon \approx 2.2204 \times 10^{-16}$). Each node consists of an Intel Xeon processor@2.4 GHz with 1 GByte of RAM. We employ a BLAS library specially tuned for this processor that achieves

Newton's method				
Step	Operation	Structure of matrices	Parallel library	Routine
1	Factorize Q	All dense	ScaLAPACK	`p_potrf`
2	Factorize R	All dense	ScaLAPACK	`p_potrf`
3	Compute $\bar{Q}^T C$	All dense	PBLAS	`p_gemm`
4.1	Solve $E^{-1}B$	Sparse E/ dense (BR^{-1})	MUMPS or ScaLAPACK	`_mumps_c` or `p_gbsv`
4.2	Solve $((E^{-1}B)\bar{R}^{-T})\bar{R}^{-1}$	All dense	PBLAS	`p_trsm`$\times 2$
5	Compute $\hat{R}_j(\hat{R}_j^T \bar{B})$	All dense	PBLAS	`p_gemm`$\times 2$
6	Compute $\bar{R}^T K_j^T$	All dense	PBLAS	`p_gemm`
7	Solve Lyapunov eq.	Sparse E, A/ dense B, K_j	SpaRed	LR-ADI iter.

LR-ADI iteration				
Step	Operation	Structure of matrices	Parallel library	Routine
1	Compute V_0	Sparse E, A/ dense $B, K, \hat{C}$	SpaRed	SMW formula
3	Compute V_{l+1}	Sparse E, A/ dense B, K, V_l	SpaRed	SMW formula

SMW formula				
Step	Operation	Structure of matrices	Parallel library	Routine
1	Solve $\bar{A}^{-T}W$	Sparse $\bar{A}$/ dense W	MUMPS or ScaLAPACK	`_mumps_c` `p_gbsv`
2	Solve $\bar{A}^{-T}K$	Sparse $\bar{A}$/ dense K	MUMPS or ScaLAPACK	`_mumps_c` `p_gbsv`
3	Compute $I_m - B^T T$	All dense	PBLAS	`p_gemm`
4	Solve TF^{-1}	All dense	ScaLAPACK	`p_gesv`
5	Compute $V + T(B^T V)$	All dense	PBLAS	`p_gemm`$\times 2$

Table 1. Parallelization of the major matrix operations that appear in the ARE solver.

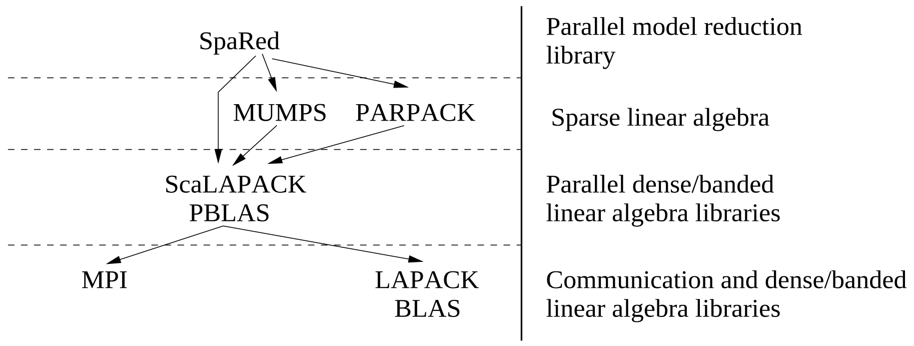
SpaRed
MUMPS
PARPACK
ScaLAPACK
PBLAS
MPI
LAPACK
BLAS
Parallel model reduction library
Sparse linear algebra
Parallel dense/banded linear algebra libraries
Communication and dense/banded linear algebra libraries

Fig. 1. Multilayered architecture of the ARE solver.

around 3800 MFLOPs (millions of flops per second) for the matrix product (routine `DGEMM` from Goto BLAS, `http://www.tacc.utexas.edu/resources/software/`). The nodes are connected via a *Myrinet* multistage network and the MPI communication library is specially developed and tuned for this network. The performance of the interconnection network was measured by a simple loop-back message transfer resulting in a latency of 18 μsec. and a bandwidth of 1.4 Gbit/s.

In order to evaluate the performance of our ARE solvers we employ two different examples:

Example 1. This standard system is obtained from a finite difference discretization (with equidistant grid) of a 2-D heat equation. The dimension of the system is given by the number of grid points, n_0, in one direction, so that $n = n_0^2$. The system is single-input, single-output (SISO); that is, $m = p = 1$.

Example 2. This model arises in a manufacturing method for steel profiles [21, 25]. The goal is to design a control that achieves moderate temperature gradients when the rail is cooled down. The mathematical model corresponds to the boundary control for a 2-D heat equation. A finite element discretization, followed by adaptive refinement via bisection results in a generalized systems of order n=79841 with 7 inputs and 6 outputs.

In both examples we employ weight matrices $Q = I_p$ and $R = I_m$.

We next report the execution times of the sparse ARE solver in Table 2 using $n_p = 16$ processing nodes. For ScaLAPACK a logical 4×4 grid was selected with the distribution block size equal 32. Other logical topologies/block sizes did not offer significative differences. A modest number of shifts is used for the LR-ADI iteration due to the need of storing the LU factors in the current implementation of the algorithm. We could overcome this restriction by recomputing the factorization at each iteration. However, doing so would produce a notable raise in the execution time. A different approach would be that of maintaining the factors stored on disk. The LR-ADI iteration was stopped after 100 iterations for Example 2. Although more iterations were strictly necessary according to the convergence criterion employed for the LR-ADI iteration, stopping the iteration at this point for this particular example guaranteed Newton's method to converge to the desired stabilizing solution of the equation.

	n	#iter. Newton	#shifts LR-ADI	Avg. #iter. LR-ADI	r	Ex. time
Example 1	160000	11	10	80	160	1h 25m 5s
Example 2	79841	5	20	100	1300	1h 4m 30s

Table 2. Execution times of the ARE solver on n_p=16 nodes.

The parallel performance of the ARE solver is highly dependent on the efficacy of the underlying parallel sparse linear system solver and the sparsity pattern of the matrix pair (A, E). Due to the problem dimensions and structure, the number of shifts selected for each example, the numerical tools that were employed (MUMPS for the solution of the sparse linear systems in both cases), and the specifications of the hardware resources (1 Gbyte of RAM per node), virtual memory was required to solve the problems using less than n_p=16 nodes; in those cases where the problem could still be solved using storage on disk, I/O resulted in much larger execution times.

6 Concluding Remarks

We have presented a solver for large-scale generalized algebraic Riccati equations with sparse matrix pair (A, E). The method involves the solution of large-scale linear systems, with sparse coefficient matrix, and well-known dense linear algebra operations on small-scale matrices. These operations are available in parallel linear algebra libraries such as ScaLAPACK, PLAPACK, MUMPS, SuperLU, etc. The experimental results on a cluster of moderate dimensions illustrates a parallel efficacy that enables the solution of equations with dimension up to $\mathcal{O}(10^5)$ in a relatively short time. Using these parallel algorithms, the solution of large-scale optimal control problems and model reduction of large-scale systems via relative error methods become thus feasible.

Acknowledgments

This research has been partially supported by the DAAD programme Acciones Integradas HA2005-0081. José M. Badía, Rafael Mayo, and E.S. Quintana-Ortí were supported by the CICYT project TIN2005-09037-C02-02 and FEDER, and project No. P1B-2004-6 of the *Fundación Caixa-Castellón/Bancaixa and UJI.*

References

1. J. Abels and P. Benner. CAREX – a collection of benchmark examples for continuous-time algebraic Riccati equations (version 2.0). SLICOT Working Note 1999-14, November 1999. Available from `http://www.slicot.org`.
2. P.R. Amestoy, I.S. Duff, J. Koster, and J.-Y. L'Excellent. MUMPS: a general purpose distributed memory sparse solver. In *Proc. PARA2000, 5th International Workshop on Applied Parallel Computing*, pages 122–131, 2000.
3. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, third edition, 1999.
4. A.C. Antoulas. *Approximation of Large-Scale Dynamical Systems.* SIAM Publications, Philadelphia, PA, 2005.
5. J.M. Badía, P. Benner, R. Mayo, and E.S. Quintana-Ortí. Balanced truncation model reduction of large and sparse descriptor linear systems. In preparation.
6. P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving Linear-Quadratic Optimal Control Problems on Parallel Computers. Preprint SFB393/05-19, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG, March 2006. Available from `http://www.mathematik.tu-chemnitz.de/preprint/2005/SFB393_19.html`.
7. P. Benner and R. Byers. An exact line search method for solving generalized continuous-time algebraic Riccati equations. *IEEE Trans. Automat. Control*, 43(1):101–107, 1998.
8. L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, PA, 1997.
9. C.-K. Cheng, J. Lillis, S. Lin, and N.H. Chang. *Interconnect Analysis and Synthesis.* John Wiley & Sons, Inc., New York, NY, 2000.
10. B.N. Datta. Linear and numerical linear algebra in control theory: Some research problems. *Linear Algebra Appl.*, 197/198:755–790, 1994.
11. R. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 12:267–319, 2003.
12. R. W. Freund and P. Feldmann. Reduced-order modeling of large passive linear circuits by means of the SyPVL algorithm. In *Technical Digest of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 280–287. IEEE Computer Society Press, 1996.
13. M. Gawronski. *Balanced control of flexible structures*, volume 211 of *Lecture Notes in Control and Information Sciences.* Springer-Verlag, London, UK, 1996.
14. M. Gawronski. *Dynamics and control of structures: A modal approach.* Mechanical Engineering Series. Springer-Verlag, Berlin, FRG, 1998.
15. M. Green. Balanced stochastic realization. *Linear Algebra Appl.*, 98:211–247, 1988.
16. D.L. Kleinman. On an iterative technique for Riccati equation computations. *IEEE Trans. Automat. Control*, AC-13:114–115, 1968.
17. P. Lancaster and L. Rodman. *The Algebraic Riccati Equation.* Oxford University Press, Oxford, 1995.
18. J.-R. Li and J. White. Low rank solution of Lyapunov equations. *SIAM J. Matrix Anal. Appl.*, 24(1):260–280, 2002.
19. V. Mehrmann. *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution.* Number 163 in Lecture Notes in Control and Information Sciences. Springer-Verlag, Heidelberg, July 1991.

20. R. Ober. Balanced parametrizations of classes of linear systems. *SIAM J. Cont. Optim.*, 29:1251–1287, 1991.
21. T. Penzl. Algorithms for model reduction of large dynamical systems. *Linear Algebra Appl.*, 415:322–343, 2006. (Reprint of Preprint SFB393/99-40, TU Chemnitz, 1999.)
22. T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 2000.
23. T. Penzl. LYAPACK Users Guide. Preprint SFB393/00-33, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG, 2000. Available from `http://www.tu-chemnitz.de/sfb393/sfb00pr.html`.
24. V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, NY, 1996.
25. F. Tröltzsch and A. Unger. Fast solution of optimal control problems in the selective cooling of steel. Preprint SFB393/99-7, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG, 1999. Available from `http://www.tu-chemnitz.de/sfb393/sfb99pr.html`.
26. E.L. Wachspress. The ADI model problem, 1995. Available from the author.