Int. J. Appl. Math. Comput. Sci., 2001, Vol.11, No.5, 1123–1150

EFFICIENT NUMERICAL ALGORITHMS FOR BALANCED STOCHASTIC TRUNCATION[†]

Peter BENNER[*], Enrique S. QUINTANA-ORTÍ[**]
Gregorio QUINTANA-ORTÍ[**]

We propose an efficient numerical algorithm for relative error model reduction based on balanced stochastic truncation. The method uses full-rank factors of the Gramians to be balanced versus each other and exploits the fact that for large-scale systems these Gramians are often of low numerical rank. We use the easy-to-parallelize sign function method as the major computational tool in determining these full-rank factors and demonstrate the numerical performance of the suggested implementation of balanced stochastic truncation model reduction.

Keywords: model reduction, stochastic realization, balanced truncation, sign function method, Newton's method

1. Introduction

Consider the Transfer Function Matrix (TFM) $G(s) = C(sI - A)^{-1}B + D$ and the associated stable but not necessarily minimal realization of a linear time-invariant (LTI) system,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), \quad t \geq 0 \end{aligned} \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$. We assume that A is stable, i.e., the spectrum of A, denoted by $\Lambda(A)$, is contained in the open left half plane. This implies that the system (1) is stable, that is, all the poles of $G(s)$ have strictly negative real parts. The number of state variables, n, is said to be the order of the system. Moreover, for the method considered in this paper we need to assume that $0 < p \leq m$ and rank $(D) = p$, which implies that $G(s)$ must not be strictly

[†] Partially supported by the DAAD programme *Acciones Integradas Hispano-Alemanas* and *Conselleria de Cultura y Educación de la Generalitat Valenciana GV99–59–1–14.*

[*] Zentrum für Technomathematik, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D-28334 Bremen, Germany, e-mail: benner@math.uni-bremen.de

[**] Departamento de Informática, Universidad Jaume I, 12080 Castellón, Spain, e-mail: {quintana,gquintan@inf.uji.es}

proper. For strictly proper systems, the method can be applied by introducing an ϵ-regularization by adding an artificial matrix $D = \begin{bmatrix} \epsilon I_p & 0 \end{bmatrix}$ (Glover, 1986).

Here we are interested in finding a reduced-order LTI system,

$$\begin{aligned} \dot{x}_r(t) &= A_r x_r(t) + B_r u_r(t), \quad t > 0, \quad x_r(0) = x_r^0, \\ y_r(t) &= C_r x_r(t) + D_r u_r(t), \quad t \geq 0 \end{aligned} \tag{2}$$

of order r, $r \ll n$ such that the TFM $G_r(s) = C_r(sI_r - A_r)^{-1}B_r + D_r$ approximates $G(s)$.

One way to distinguish between different model reduction techniques is to specify in which way the reduced-order model approximates the original system. Absolute error methods try to minimize $\|\Delta_a\| := \|G - G_r\|$ for some system norm. The most popular methods in this area are related to balanced realizations (Moore, 1981). These methods yield *a priori* bounds on the $\mathcal{H}_\infty$-norm of Δ_a which is defined as

$$\|H\|_\infty = \operatorname*{ess\,sup}_{\omega \in \mathbb{R}} \sigma_{\max}\big(H(\jmath\omega)\big), \tag{3}$$

where H is a stable, rational matrix function, $\jmath := \sqrt{-1}$, and $\sigma_{\max}(M)$ is the largest singular value of the matrix M. That is, basically, $\|H\|_\infty$ is the "maximum" of the spectral norms of the matrices $H(\jmath\omega)$ evaluated at each *frequency* ω. Hence, $\|\Delta_a\|_\infty$ only gives the worst-case deviation of the reduced-order model from the original system. There is no information on *where* this maximum deviation occurs available. For example, balanced truncation (Moore, 1981; Tombs and Postlethwaite, 1987) tends to approximate high frequencies very well ($\lim_{\omega\to\infty} \Delta_a(\jmath\omega) = 0$), while singular perturbation approximation (Liu and Anderson, 1986) has zero steady state error, i.e., $G(0) = G_r(0)$, and good approximation properties at low frequencies.

It is often desirable that the reduced-order systems have uniform approximation properties over the whole frequency range $0 \leq \omega \leq \infty$ or give a particularly good approximation at prescribed frequencies. For example, this is the case if the LTI system describes a high-order controller that should perform well at practically relevant frequencies. This requirement can be satisfied by frequency-weighted model reduction methods and, as a special case, by relative error methods. Relative error methods attempt to minimize the relative error $\|\Delta_r\|_\infty$, defined implicitly by $G - G_r = G\Delta_r$. Among these, the balanced stochastic truncation (BST) methods (Desai and Pal, 1984; Green, 1988a; Varga and Fasol, 1993) and their relatives are particularly popular. Due to their computational cost and the involved calculations, these methods have been used so far only for systems of modest size, i.e., models of state-space dimension up to order $n = 100$. With the recently proposed Fortran 77 implementation in the *Subroutine Library in Control Theory* – SLICOT[1] (Varga, 1999), systems with a couple of hundreds of state-space variables may be treated on modern desktop computers. Here we focus on a numerically reliable and efficient implementation of BST which can be applied to systems with up to a couple of thousands of states using high-performance computing. It should be noted, though, that this method can also

[1]Available from `http://www.win.tue.nl/niconet/NIC2/slicot.html`

be applied to and is very efficient for problems of smaller size. Note that the state transition matrix A is considered to be a dense matrix; an algorithm that exploits sparsity for a BST method is currently under development.

BST is a model reduction method based on truncating a balanced stochastic realization. Such a realization is obtained as follows. Define $\Phi(s) = G(s)G^T(-s)$, and let W be a *square minimum phase right spectral factor* of Φ, satisfying $\Phi(s) = W^T(-s)W(s)$. As D has full row rank, $E := DD^T$ is positive definite and a minimal state-space realization (A_W, B_W, C_W, D_W) of W is given by (Anderson, 1967a; 1967b)

$$A_W = A, \quad B_W = BD^T + P_G C^T, \quad C_W = E^{-\frac{1}{2}}(C - B_W^T X_W), \quad D_W = E^{\frac{1}{2}}.$$

Here P_G is the controllability Gramian of $G(s)$ given by the solution of the Lyapunov equation

$$AP + PA^T + BB^T = 0, \tag{4}$$

while X_W is the observability Gramian of $W(s)$ obtained as the *stabilizing* solution of the algebraic Riccati equation (ARE)

$$(A - B_W E^{-1} C)^T X + X(A - B_W E^{-1} C) + X B_W E^{-1} B_W^T X + C^T E^{-1} C = 0. \tag{5}$$

That is, for this particular solution,

$$\hat{A} := A - B_W E^{-1} C + B_W E^{-1} B_W^T X_W \tag{6}$$

is stable. The Gramians P_G, X_W are symmetric positive (semi-)definite and admit decompositions $P_G = S^T S$ and $X_W = R^T R$, which are usually referred to as Cholesky decompositions (which they are in a strict sense only if P_G and X_W are nonsingular; see (Golub and Van Loan, 1996)). As in the computation of balanced realizations in (Moore, 1981; Tombs and Postlethwaite, 1987), a state-space transformation T can be obtained either from the dominant left and right invariant subspaces of $P_G X_W$ or the dominant left and right singular subspaces of SR^T such that the transformed system $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}) = (T^{-1}AT, T^{-1}B, CT, D)$ has a controllability Gramian $\tilde{P}_G$ satisfying

$$\tilde{P}_G := T^{-1} P_G T^{-T} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) = T^T X_W T =: \tilde{X}_W, \tag{7}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. That is, the observability Gramian $\tilde{X}_W$ of the transformed realization of the right spectral factor of $\Phi(s)$ equals the controllability Gramian of the transformed realization of $G(s)$. Such a realization of $G(s)$ is called a *balanced stochastic realization* (BSR). A model reduction based on BSR is then obtained by truncating the realization $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ to order r where $\sigma_r \gg \sigma_{r+1}$. The properties of balanced stochastic truncation (BST) are summarized in the following theorem, which collects results from (Desai and Pal, 1984; Green, 1988a; 1988b).

Theorem 1. *If $G(s) = C(sI_n - A)^{-1}B + D$ with A stable and D nonsingular is the TFM of a square LTI system, and*

$$(T^{-1}AT, T^{-1}B, CT, D) := \left(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \end{bmatrix}, D \right)$$

is a BSR such that (7) holds, then

$$(A_r, B_r, C_r, D_r) := (A_{11}, B_1, C_1, D)$$

is a stable, minimal BSR with the following properties:

(i) $G_r(s) := C_r(sI_r - A_r)^{-1}B_r + D_r$ *satisfies the relative error bound*

$$\|\Delta_r\|_\infty = \|G^{-1}(G - G_r)\|_\infty \leq \prod_{j=r+1}^{n} \frac{1+\sigma_j}{1-\sigma_j} - 1. \tag{8}$$

(ii) If $G(s)$ *is minimum-phase, i.e.,* G *has no zeros in* $\mathbb{C}^+$*, then* $G_r(s)$ *is minimum-phase.*

BST methods differ in the way the reduced-order model is actually computed. Here we focus on square-root methods, that is, SR^T is used rather than $P_G X_W$. This has the advantage of not squaring the condition number of the matrix. In (Safonov and Chiang, 1988), two implementations of BST are proposed based on (7). The first one is not a square-root method but it uses $P_G X_W$. The reduced-order model obtained with this approach is not balanced, but it satisfies the same error bound (8). It has the advantage of avoiding the computation of the possibly ill-conditioned transformation T for highly stochastically unbalanced realizations. The second approach is a square-root method yielding a stochastically balanced reduced-order model. Both ideas can be combined to yield a balancing-free square-root BST method, see (Varga and Fasol, 1993). Often, this yields transformations that are much better conditioned than for the square-root approach in (Safonov and Chiang, 1988), and working with $P_G X_W$ is avoided. Here we will consider both square-root methods where the stochastically balancing square-root approach should only be used if a stochastically balanced reduced-order model is explicitly required.

Our algorithms differ in several ways from the ones considered in (Safonov and Chiang, 1988; Varga and Fasol, 1993), though they are mathematically equivalent. In particular, we focus on the implementation on modern computer architectures such as distributed memory parallel computers. We combine the results obtained in (Benner and Byers, 1998; Benner and Quintana-Ortí, 1999; Benner *et al.*, 1999; Benner *et al.*, 2000a; 2000b) in order to derive new algorithms for BST model reduction in the following way. The Lyapunov equation (4) is solved using a sign function-based method (Benner and Quintana-Ortí, 1999; Roberts, 1980), from which we obtain a full-rank factorization $P_G = S^T S$, i.e., $S \in \mathbb{R}^{n_c \times n}$ where $n_c = \text{rank}\,(S) = \text{rank}\,(P_G)$. This is usually faster and more reliable than computing the Cholesky factor using Hammarling's method (Hammarling, 1982), see (Benner *et al.*, 2000b) for details. The parallelization of this algorithm on distributed-memory parallel computers is described in (Benner *et al.*, 1999). The same approach is used to compute a full-rank factor R of X_W from a stabilizing approximation $\tilde{X}_W$ to X_W as follows. This technique is also used in (Varga, 1999). Let $D = [\,\hat{D}^T\ 0\,]U$ be an LQ decomposition (Golub and Van Loan, 1996) of D. Note that $\hat{D} \in \mathbb{R}^{p \times p}$ is a square, nonsingular matrix as D has full row rank. Now set

$$H_W := \hat{D}^{-T}C, \quad \hat{B}_W := B_W \hat{D}^{-1}, \quad \hat{C} := (H_W - \hat{B}_W^T X).$$

Then the ARE (5) is equivalent to

$$0 = A^T X + X A + \hat{C}^T \hat{C}.$$

Using a computed approximation $\tilde{X}_W$ of X_W to form $\hat{C}$, the Cholesky or full-rank factor R of X_W can be computed directly from the Lyapunov equation

$$A^T(R^T R) + (R^T R)A + \hat{C}^T \hat{C} = 0 \tag{9}$$

using the method from (Benner *et al.*, 2000b) as in the case of eqn. (4). The approximation $\tilde{X}_W$ is obtained by solving (5) using Newton's method with exact line search as described in (Benner, 1997; Benner and Byers, 1998). The Lyapunov equations in each iteration step of Newton's method are again solved via a sign function-based method as described in (Benner *et al.*, 2000a). This allows an efficient implementation on modern serial and parallel computers and, as a stabilizing initial guess is readily available starting from zero, this algorithm usually outperforms the Schur vector method in both computing time and accuracy. (See (Benner *et al.*, 2000a) for details.) Here, the main goal is to show how parallel computing can be used to compute reduced-order models that can easily be handled, in a second stage, on a single-processor machine in a computer-aided control systems design (CACSD) environment.

The outline of the paper is as follows. In Section 2 we show how a BST reduced-order model can be computed from full-rank factors of the Gramians P_G and X_W. Next, in Section 3 we describe efficient and parallelizable numerical algorithms that are used to obtain the full-rank factors of the Gramians. The implementation of the suggested method on a Beowulf cluster and its numerical performance are explained in Section 4. Conclusions are given in Section 5.

2. Balanced Stochastic Truncation Using Full-Rank Factors of the Gramians

In this section we review the techniques that can be used to compute the projection matrices which determine the reduced-order model. We suggest a modification of two of the algorithms by replacing the Cholesky factors of the Gramians by full-rank factors, resulting in a smaller arithmetic cost and workspace requirements in the case of (numerically) rank-deficient Gramians.

We have noted so far that the Gramians to be balanced versus each other are obtained as the solution of the stable, nonnegative Lyapunov equation (4) and the stabilizing, positive semidefinite solution of the ARE (5). Hence, both Gramians are positive semidefinite and can be decomposed as $P_G = S^T S$ and $X_W = R^T R$. The factors $S, R \in \mathbb{R}^{n \times n}$ can be chosen triangular and are called the Cholesky factors of the Gramians.

The transformation matrix T that achieves the balancing in (7) can be obtained from S and R following the ideas of balanced truncation as demonstrated in (Laub *et al.*, 1987; Tombs and Postlethwaite, 1987). That is, BST model reduction can be achieved using SR^T instead of the product of the Gramians themselves. The resulting *square-root (SR) algorithm* avoids dealing with the Gramians as their condition

number can be up to the square of the condition number of the Cholesky factors. In these algorithms, eqns. (4) and (5) are initially solved for the Cholesky factors without ever forming the Gramians explicitly. This can be achieved, e.g., by the algorithms described in (Benner and Quintana-Ortí, 1999; Hammarling, 1982) applied to (4) and (9). Then the SVD of the product

$$SR^T = [U_1\ U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \tag{10}$$

is computed. Here the matrices are partitioned at a given dimension r, with $\Sigma_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$ and $\Sigma_2 = \operatorname{diag}(\sigma_{r+1}, \ldots, \sigma_n)$ such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} \geq \sigma_{r+2} \geq \cdots \geq \sigma_n \geq 0. \tag{11}$$

Note that the σ_k's are the Hankel singular values of the so-called *phase matrix* $(W(-s)^T)^{-1}G(s)$. To achieve a good reduced-order model, r should be chosen in order to give a natural separation of the states, i.e., one should look for a large gap in the singular values, $\sigma_r \gg \sigma_{r+1}$ (Tombs and Postlethwaite, 1987).

Notice that the SVD in (10) can be obtained without explicitly forming the product of the Cholesky factors using the techniques in (Fernando and Hammarling, 1988; Heath *et al.*, 1987). Finally, defining

$$T_{\text{left}} = \Sigma_1^{-1/2} V_1^T R, \quad T_{\text{right}} = S^T U_1 \Sigma_1^{-1/2}, \tag{12}$$

the reduced-order system is given by

$$A_r = T_{\text{left}} A T_{\text{right}}, \quad B_r = T_{\text{left}} B, \quad C_r = C T_{\text{right}}, \quad D = D_r. \tag{13}$$

As has been shown in Theorem 1, the reduced-order model is a stable balanced stochastic realization satisfying the error bound (8).

As the reduced-order model in (13) is stochastically balanced, the projection matrices in (12) tend to be ill-conditioned if the original system is highly unbalanced, resulting in inaccurate reduced-order models. An alternative here are the *balancing-free (BF) algorithms* (Safonov and Chiang, 1988). Here, the reduced-order model is not balanced. In these algorithms, after solving (4) and (5), (9) for the Gramians, an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ is computed such that

$$Q P_G X_W Q^T \tag{14}$$

is in upper real Schur form. Using an eigenvalue reordering procedure as described, e.g., in (Golub and Van Loan, 1996), a pair of orthogonal matrices $Q_a,\ Q_d \in \mathbb{R}^{n \times n}$ is computed such that the diagonal blocks in $Q_a^T P_G X_W Q_a$ and $Q_d^T P_G X_W Q_d$ are ordered, respectively, in ascending and descending order of the absolute magnitude of the eigenvalues. Let $Q_a = [Q_{a_1}, Q_{a_2}]$ and $Q_d = [Q_{d_1}, Q_{d_2}]$, with $Q_{a_2},\ Q_{d_1} \in \mathbb{R}^{n \times r}$. Then the SVD

$$Q_{a_2}^T Q_{d_1} = [U_1\ U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \tag{15}$$

provides the orthonormal matrices that allow the construction of the reduced-order system as in (12) and (13). The method can also be adapted to work on the Cholesky factors thus providing a square-root algorithm.

The *Balancing-free square-root (BFSR) algorithms* for BST combine the best characteristics of the SR and BF approaches (Varga and Fasol, 1993). BFSR algorithms share the first two steps (solving (4) and (5) for the Cholesky factors and computing the SVD in (10)) with the SR algorithms. Then orthogonal bases for the ranges of $S^T U_1$ and $R^T V_1$ are computed via the two QR factorizations

$$S^T U_1 = [P_1\ P_2] \begin{bmatrix} \tilde{S} \\ 0 \end{bmatrix}, \quad R^T V_1 = [Q_1\ Q_2] \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}, \tag{16}$$

where $P_1,\ Q_1 \in \mathbb{R}^{n\times r}$ have orthonormal columns and $\tilde{S},\ \tilde{R} \in \mathbb{R}^{r\times r}$ are upper triangular. Note that $P_2,\ Q_2$ are not needed; it is sufficient to compute the "skinny" QR decompositions $S^T U_1 = P_1 \tilde{S}$ and $R^T V_1 = Q_1 \tilde{R}$.

The reduced-order system is then given by (13), where the projection matrices are given by the formulae

$$T_{\text{left}} = (Q_1^T P_1)^{-1} Q_1^T, \quad T_{\text{right}} = P_1,$$

which take into account that $T_{\text{left}} T_{\text{right}} = I_r$ is needed.

For the implementations reported in this paper we chose the SR and BFSR algorithms as the BF algorithm usually shows no advantage over BFSR algorithms with respect to model reduction abilities. Moreover, the BF approach is potentially numerically unstable. For one, it uses the product $P_G X_W$ rather than SR^T, leading to a squaring of the condition number of the matrix product. Second, the projection matrices T_{left} and T_{right} computed by the BFSR approach are often significantly better conditioned than those computed by the BF approach (Varga and Fasol, 1993). Furthermore, both SR and BFSR algorithms can be implemented efficiently on parallel computers. On the other hand, the BF method needs a parallel implementation of the QR algorithm (see eqn. (14)) and re-ordering of eigenvalues. Both stages present severe difficulties regarding their parallel implementation; for a more detailed discussion and references, see (Benner *et al.*, 2000b). For these reasons, we avoid the implementation of the BF algorithm.

So far we have assumed that the Cholesky factors S and R of the Gramians are square $n \times n$ matrices. For non-minimal systems, we have rank $(S) < n$ and/or rank $(R) < n$. Even in the case when, theoretically, both Gramians are nonsingular, they are often numerically rank-deficient (Penzl, 2000). This can be exploited by computing rectangular factors $\tilde{S}, \tilde{R}$ of the Gramians such that $\|P_G - \hat{S}^T \hat{S}\|_F \leq \gamma_S \varepsilon$ and $\|X_W - \hat{R}^T \hat{R}\|_F \leq \gamma_R \varepsilon$, where ε is the machine precision and γ_S, γ_R are small constants.

Hence, rather than working with the Cholesky factors, we may use *full-rank factors* of $P_G,\ X_W$. As P_G and X_W are positive semidefinite, there exist matrices

$\hat{S} \in \mathbb{R}^{n_c \times n}$, $\hat{R} \in \mathbb{R}^{n_o \times n}$ such that $P_G = \hat{S}^T \hat{S}$, $X_W = \hat{R}^T \hat{R}$, and

$$n_c := \operatorname{rank}\left(\hat{S}\right) = \operatorname{rank}(S) = \operatorname{rank}(P_G),$$
$$n_o := \operatorname{rank}\left(\hat{R}\right) = \operatorname{rank}(R) = \operatorname{rank}(X_W).$$

Here, by abuse of notation, we employ $\operatorname{rank}(M)$ to denote the *numerical rank* (Golub and Van Loan, 1996) of a matrix M. The full-rank factors $\hat{S}$, $\hat{R}$ can be obtained from S and R by omitting the trailing rows of the factors that are numerically zero due to the triangular form of S and R. (For non-triangular factorizations, it is sufficient to compute QR factorizations of S and R.) A more efficient way is to compute $\hat{S}$ and $\hat{R}$ directly, e.g., with the algorithm described in Section 3.1 applied to (4) and (9), respectively. In the latter case, S and R can be defined by $S := \begin{bmatrix} \hat{S} \\ 0 \end{bmatrix}$, $R := \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$. The SVD in (10) can then be obtained from that of $\hat{S}\hat{R}^T$ as follows. Here we assume that $n_c \geq n_o$. The case $n_c < n_o$ can be treated analogously. Then we can compute the SVD

$$\hat{S}\hat{R}^T = \hat{U} \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix} \hat{V}^T, \quad \hat{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_{n_o}), \tag{17}$$

where $\hat{U} \in \mathbb{R}^{n_c \times n_c}$, $\hat{V} \in \mathbb{R}^{n_o \times n_o}$. Partitioning $\hat{U} = \begin{bmatrix} \hat{U}_1 & \hat{U}_2 \end{bmatrix}$ such that $\hat{U}_1 \in \mathbb{R}^{n_c \times n_o}$, the SVD of SR^T is given by

$$SR^T = \left[\begin{array}{c|cc} \hat{U}_1 & \hat{U}_2 & 0 \\ \hline 0 & 0 & I_{n-n_c} \end{array}\right] \left[\begin{array}{c|c} \hat{\Sigma}_1 & 0 \\ \hline 0 & 0 \end{array}\right] \left[\begin{array}{c|c} \hat{V}^T & 0 \\ \hline 0 & I_{n-n_o} \end{array}\right]. \tag{18}$$

Then the decision on the index r yielding the size of the reduced-order model can be based on the singular values of $\hat{\Sigma}_1$. If we use r to partition $\hat{U}_1 = \begin{bmatrix} \hat{U}_{11} & \hat{U}_{12} \end{bmatrix}$, $\hat{V} = \begin{bmatrix} \hat{V}_1 & \hat{V}_2 \end{bmatrix}$, and $\hat{\Sigma}_1 = \begin{bmatrix} \hat{\Sigma}_{11} & 0 \\ 0 & \hat{\Sigma}_{12} \end{bmatrix}$ with $\hat{U}_{11} \in \mathbb{R}^{n_c \times r}$, $\hat{V}_1 \in \mathbb{R}^{n_o \times r}$, and $\hat{\Sigma}_{12} \in \mathbb{R}^{r \times r}$, then the subsequent computations can be performed working with $\hat{U}_{11}$, $\hat{\Sigma}_{11}$, and $\hat{V}_1$ rather than using the data from the full-size SVD in (18). This can easily be seen by replacing S, R with $\hat{S}, \hat{R}$ and U_1, V_1 with $\hat{U}_{11}, \hat{V}_1$ in (12) and (16). This leads to significant savings of workspace and computational cost. For example, using the Golub–Reinsch SVD (Golub and Van Loan, 1996), (10) requires $22n^3$ flops (floating–point arithmetic operations) and workspace for $2n^2$ real numbers (note that we need to form U, V explicitly), while (17) requires only $14n_c n_o^2 + 8n_o^3$ flops and workspace for $n_c^2 + n_o^2$ real numbers. In particular, for large-scale dynamical systems, the *numerical rank* of P_G, X_W and $\hat{S}$, $\hat{R}$ is often much smaller than n, see (Penzl, 1999; 2000). Suppose that (numerically) $n_c = n_o = n/10$ (which can quite frequently be observed when the system comes from the semi-discretization of parabolic or hyperbolic partial differential equations). Then the computation of (17) is 1000 times less expensive than that of (10), and only 1% of the workspace is required for (17) as compared to (10). Some more savings are obtained from the cheaper computation of the projection matrices yielding the reduced-order models.

In the next section we show how the full-rank factors $\hat{S}$ and $\hat{R}$ can be computed directly without having to compute S, R or even P_G, X_W first.

3. Computing the Gramians

The following key idea will be used here: if the controllability Gramian P_G of $G(s)$ and the observability Gramian X_W of $W(s)$ are (numerically) rank-deficient, positive semidefinite matrices, we can find rectangular matrices $\hat{S} \in \mathbb{R}^{n_c \times n}$, $\hat{R} \in \mathbb{R}^{n_o \times n}$ such that $\hat{S}^T\hat{S}$ and $\hat{R}^T\hat{R}$ equal (up to numerical accuracy) P_G and X_W, respectively. Therefore, we need an algorithm that solves the stable, nonnegative Lyapunov equations (4) and (9) for a full-rank factor of the solution directly. Furthermore, we need an algorithm for solving the ARE (5). As our focus is on high-performance computing and particularly on parallel algorithms for distributed-memory architectures, our methods for solving Lyapunov equations will be based on the sign function method which is known to be easily and efficiently parallelizable (Bai and Demmel, 1993; Bai *et al.*, 1997; Benner *et al.*, 1999; Gardiner and Laub, 1991).

A sign function-based Lyapunov solver is then also used in the inner loop of Newton's method for solving AREs; we use the same procedure to solve (5).

3.1. Solving Stable Lyapunov Equations with the Sign Function Method

In this section we describe Lyapunov equation solvers based on the matrix sign function. These are specially appropriate for parallel distributed memory computers. Details of the algorithms can be found in (Benner and Quintana-Ortí, 1999). Some minor modifications of the algorithms in order to address some properties of the Lyapunov equations corresponding to model reduction are described here on the analogy of (Benner *et al.*, 2000b).

Consider a matrix $Z \in \mathbb{R}^{n\times n}$ with no eigenvalues on the imaginary axis and Jordan decomposition $Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1}$. Here, the Jordan blocks in $J^- \in \mathbb{R}^{k\times k}$ and $J^+ \in \mathbb{R}^{(n-k)\times(n-k)}$ contain, respectively, the eigenvalues of Z in the open left and right half planes. The *matrix sign function* of Z is then defined as $\operatorname{sign}(Z) := S \begin{bmatrix} -I_k & 0 \\ 0 & I_{n-k} \end{bmatrix} S^{-1}$, where I_k denotes the identity matrix of order k. Note that $\operatorname{sign}(Z)$ is unique and independent of the order of the eigenvalues in the Jordan decomposition of Z; see, e.g., (Lancaster and Rodman, 1995). Many other definitions of the sign function can be given; see (Kenney and Laub, 1995) for an overview.

The matrix sign function was shown to be useful in many problems involving spectral decomposition as $(I_n - \operatorname{sign}(Z))/2$ defines the skew projector onto the stable Z-invariant subspace parallel to the unstable subspace. (By the *stable* invariant subspace of Z we denote the Z-invariant subspace corresponding to the eigenvalues of Z in the open left half plane.)

Applying Newton's root-finding iteration to $Z^2 = I_n$, where the starting point is chosen as Z, we obtain the Newton iteration for the matrix sign function:

$$Z_0 \leftarrow Z, \quad Z_{k+1} \leftarrow \frac{1}{2}(Z_k + Z_k^{-1}), \quad k = 0, 1, 2, \dots \ . \tag{19}$$

Under the given assumptions for the spectrum of Z, the sequence $\{Z_k\}_{k=0}^{\infty}$ converges to $\operatorname{sign}(Z)$ (Roberts, 1980).

Although the convergence of the Newton iteration is globally quadratic, the initial convergence may be slow. Acceleration is possible, e.g., via *determinantal scaling* (Byers, 1987),

$$Z_k \leftarrow c_k Z_k, \quad c_k = |\det(Z_k)|^{-\frac{1}{n}},$$

where $\det(Z_k)$ denotes the determinant of Z_k. Other acceleration schemes can be employed; see (Bai and Demmel, 1993) for a comparison of these schemes.

Roberts (1980) was the first to use the matrix sign function for solving Lyapunov (and Riccati) equations. In the proposed method, the solution of the stable Lyapunov equation

$$F^T X + XF + Q = 0 \tag{20}$$

is computed by applying the Newton iteration (19) to the Hamiltonian matrix $H = \begin{bmatrix} F & 0 \\ Q & -F^T \end{bmatrix}$ associated with (20). The solution matrix X_* can then be determined from the stable Z–invariant subspace given by the range of the projector $(I_n - \operatorname{sign}(H))/2$. Roberts also shows that, when applied to H, the Newton iteration (19) can be simplified to

$$\begin{aligned} F_0 &\leftarrow F, \quad F_{k+1} \leftarrow \frac{1}{2}\left(F_k + F_k^{-1}\right), \\ Q_0 &\leftarrow Q, \quad Q_{k+1} \leftarrow \frac{1}{2}\left(Q_k + F_k^{-T} Q_k F_k^{-1}\right), \end{aligned} \qquad k = 0, 1, 2, \ldots \tag{21}$$

and $X_* = \frac{1}{2}\lim_{k\to\infty} Q_k$. The sequences for F_k and Q_k require $6n^3$ flops per iteration so that 5–6 iterations are as expensive as the Bartels–Stewart method (Bartels and Stewart, 1972).

Iteration (21) can be applied to (4) by setting $F = A^T$ and $Q_0 := BB^T$ in order to obtain the controllability Gramian as $P_G = Q_\infty/2$. At convergence, $P_G = Q_\infty/2$.

As F is a stable matrix, $F_\infty = \lim_{k\to\infty} F_k = -I_n$, and a suitable convergence criterion for the iterations is to stop when the relative error in the sequence of F_k drops below a tolerance threshold, i.e., if

$$\|F_k + I_n\|_\infty \le \tau \|F_k\|_\infty,$$

for a user-defined tolerance τ. In our implementations we employ $\tau = n\sqrt{\varepsilon}$ and perform two additional iterations once the convergence criterion is satisfied. Due to the quadratic convergence of the Newton iteration, this is usually enough to reach the attainable accuracy.

In (Benner and Quintana-Ortí, 1999; Larin and Aliev, 1993), iteration (21) was modified to obtain the Cholesky factors rather than the solutions themselves. The basic idea is that if we can factor $Q = BB^T$ and set $B_0 = B$, the iterations for the symmetric matrices Q_k can be written in a factored form as

$$Q_{k+1} = B_{k+1} B_{k+1}^T = \frac{1}{2c_k} \begin{bmatrix} B_k & c_k F_k^{-T} B_k \end{bmatrix} \begin{bmatrix} B_k^T \\ c_k B_k^T F_k^{-1} \end{bmatrix},$$

yielding an iteration of the form

$$B_{k+1} \leftarrow \frac{1}{\sqrt{2c_k}} \begin{bmatrix} B_k & c_k F_k^{-T} B_k \end{bmatrix}. \tag{22}$$

Using (22), the workspace required to store the sequence of B_k is doubled in each iteration step. This can be avoided by computing in each iteration step an LQ factorization of $B_{k+1}B_{k+1}^T$ such that

$$B_{k+1} = \begin{bmatrix} S_{k+1} & 0 \end{bmatrix} U_{k+1}.$$

As

$$B_k B_k^T = S_k S_k^T,$$

it is sufficient to store the triangular factors in B_k. The orthogonal factors need not be accumulated, but still the amount of work required in each iteration step is increased by these factorizations. Therefore, in (Benner and Quintana-Ortí, 1999), a compromise is proposed: first, fix the available workspace for the sequence of B_k. A reasonable amount is an $n \times 2n$ array as the rank of the required Cholesky factor S cannot exceed n. Therefore, we may use (22) as long as $B_k \in \mathbb{R}^{n \times m_k}$, with $m_k \leq n$, and then use the triangular factors obtained by the LQ factorizations. Hence, we perform $k \leq \log_2 \frac{n}{m}$ iterations with (22) before starting to compute factorizations in each step and only use the triangular factors. If convergence is achieved before the switch, we have to compute a final LQ factorization to obtain the Cholesky factor.

Here we propose a slightly different strategy taking into account that for large-scale systems the Cholesky factors are often of low numerical rank and therefore we can save some workspace and arithmetic work by not allowing the iterates for the Cholesky factors to become rank-deficient. That is, in each step we compute a rank-revealing LQ decomposition of the matrix in (22), using an LQ decomposition with row pivoting (Golub and Van Loan, 1996). The rank of the iterates is then determined using an incremental condition estimator (Bischof, 1990) resulting in the following decomposition:

$$\frac{1}{\sqrt{2c_k}} \begin{bmatrix} B_k & c_k F_k^{-1} B_k \end{bmatrix} = \Pi_{k+1}^B \begin{bmatrix} (S_{k+1})_{11} & 0 \\ (S_{k+1})_{21} & (S_{k+1})_{22} \end{bmatrix} U_{k+1}.$$

Here U_{k+1} is orthogonal, Π_{k+1}^B is a permutation matrix, $(S_{k+1})_{11} \in \mathbb{R}^{s_{k+1} \times s_{k+1}}$ is lower triangular with s_{k+1} being the estimated rank of the iterate, and $(S_{k+1})_{22} \approx 0$. Setting $(S_{k+1})_{22} = 0$, we can proceed with the new iterate

$$B_{k+1} \leftarrow \Pi_{k+1}^B \begin{bmatrix} (S_{k+1})_{11} \\ (S_{k+1})_{21} \end{bmatrix} \in \mathbb{R}^{n \times s_{k+1}}.$$

If $X = \hat{Y}\hat{Y}^T$ is a full-rank factorization of the solution X of (20), then it follows that

$$\frac{1}{\sqrt{2}} \lim_{k \to \infty} B_k = \hat{Y}.$$

If applied to (4) with $B_0 = B$ and $F_0 = A^T$, it follows that the sequence $\{\frac{1}{\sqrt{2}}B_k\}_{k=0}^{\infty}$ converges to the full-rank factor $\hat{S}^T$ of the controllability Gramian. In our experiments reported in Section 4 we base our rank decisions on a relative tolerance threshold $10n\varepsilon$, where n is the size of the matrix.

As in most applications $m \ll n$ and the *numerical rank* (see, e.g., (Golub and Van Loan, 1996)) of the Cholesky factor S of the controllability Gramian is also usually much smaller than n, this technique quite often saves a large amount of workspace and computational cost compared to using the technique described above where the workspace for the iterates is increased up to size $n \times n$. As outlined in Section 2, the product $\hat{S}\hat{R}^T$ is a small size, in general rectangular matrix for which the subsequent computations needed for model reduction are much cheaper than for the full or triangular $n \times n$ Cholesky factors as obtained by Hammarling's method, used in the implementation of the BST model reduction algorithms described in (Varga, 1999; Varga and Fasol, 1993).

The Lyapunov solvers described above are iterative procedures composed of LU and QR factorizations, triangular linear systems, and matrix inversions. These matrix operations are of wide appeal for computer architectures that can take advantage of block-partitioned algorithms, and specially for parallel distributed architectures (Blackford *et al.*, 1997; Dongarra *et al.*, 1986).

Remark 1. By allowing $\|(S_{k+1})_{22}\|$ to become larger or by fixing the largest number of allowed columns in S_k, the above procedure can also be used to obtain low-rank approximations of the Cholesky or full-rank factors. In particular, for LTI systems having Gramians with fast decaying eigenvalues such low-rank approximations often yield all relevant information needed for model reduction. This is also used in (Penzl, 1999) for the model reduction algorithms developed there for LTI systems with a sparse coefficient matrix A.

3.2. Solving Algebraic Riccati Equations with an Exact Line Search Method

Kleinman showed (1968) that under suitable conditions Newton's method applied to the standard ARE converges to the desired stabilizing solution of the ARE. Variants of Newton's method for solving ARE are presented in (Benner and Byers, 1998; Lancaster and Rodman, 1995).

Usually, Newton's method for AREs is formulated for the ARE arising in linear-quadratic optimal control, i.e.,

$$0 = Q + F^T X + XF - XGX, \tag{23}$$

where $G, Q \in \mathbb{R}^{n\times n}$ are symmetric positive semidefinite. In the case of the ARE (5) to be considered here, G is negative semidefinite. It is not difficult to see that all the convergence results for Newton's method applied to (23) can be derived in a similar

way for the case considered here; see (Benner, 1997; Varga, 1995). We use these results here to formulate Newton's method for

$$0 = Q + F^T X + XF + XGX =: \mathcal{R}(X) \tag{24}$$

with G positive semidefinite. This can then be applied to (5) with

$$F := A - B_W E^{-1} C, \quad G := B_W E^{-1} B_W^T, \quad Q := C^T E^{-1} C. \tag{25}$$

Although Newton's method for AREs converges ultimately quadratically from any starting guess X_0 such that $A + GX_0$ is stable (see Theorem 2), initial convergence may be slow. Even worse, sometimes the first step is an enormous leap away from X_0 and X_* and only returns slowly afterwards. Therefore, we use modified Newton's method employing an exact line search technique in order to accelerate convergence in early stages of the iteration. This technique was proposed for (23) in (Benner and Byers, 1998) and for (24) in (Benner, 1997).

The modified Newton iteration for (24) can be formulated as follows.

Algorithm 1. (Modified Newton's method for the ARE).
Input: $F, G, Q \in \mathbb{R}^{n\times n}$, $G = G^T$, $Q = Q^T$, $X_0 = X_0^T$ — *an initial guess.*
Output: *Approximate solution* $X_{j+1} \in \mathbb{R}^{n\times n}$ *of (24).*

`FOR` $j = 0, 1, 2, \ldots$ "until convergence"

1. $F_j = F + GX_j$.
2. Solve the Lyapunov equation

$$0 = \mathcal{R}(X_j) + F_j^T N_j + N_j F_j$$

   for N_j.
3. Compute the line search parameter t_j.
4. $X_{j+1} = X_j + t_j N_j$.

`END FOR`

Standard Newton's method for the ARE is recovered from the above algorithm by setting $t_j = 1$ for all j.

In our implementation, we employ the sign function-based method (21) to solve the Lyapunov equations in each iteration step in order to determine the Newton step N_j. As this is the major computational step in Newton's method for solving (5), this algorithm can be efficiently implemented on parallel computers; see the discussion at the end of Subsection 3.1 and (Benner *et al.*, 2000a).

Possible stopping criteria for this algorithm are discussed in (Benner *et al.*, 2000a). Roughly speaking, the iteration can be stopped once $\|\mathcal{R}(X_j)\| \leq \tau \|X_j\|$ or $\|N_j\| \leq \tau \|X_j\|$ for some user-defined tolerance threshold τ.

The line search parameter t_j is determined as a local minimizer of the one-dimensional optimization problem to minimize the residual of the next step with respect to t_j, i.e., to find a local minimum of

$$\begin{aligned} f_j(t) &= \|\mathcal{R}(X_{j+1})\|_F^2 = \operatorname{trace}\left(\mathcal{R}(X_j + tN_j)^2\right) \\ &= \operatorname{trace}\left(\mathcal{R}(X_j)^2\right)(1-t)^2 - 2\operatorname{trace}\left(\mathcal{R}(X_j)V_j\right)(1-t)t^2 \\ &\quad + \operatorname{trace}\left(V_j^2\right)t^4, \end{aligned} \tag{26}$$

where $V_j = N_j G N_j$. In (Benner, 1997) it is proved that such a minimizing t_j exists in $[0,2]$ such that

$$\|\mathcal{R}(X_j + t_j N_j)\|_F \leq \|\mathcal{R}(X_j + N_j)\|_F.$$

The restriction to the interval $[0,2]$ is necessary in order to ensure that the X_j remain stabilizing; see (Benner, 1997; Benner and Byers, 1998). Although the line search increases the cost of Newton's method (mainly because of the computation of V_j), in practice, this overhead is largely compensated by a reduction in the number of iterations of Newton's method and therefore in the overall cost of the solver.

Moreover, in large-scale applications as considered here the additional cost caused by the line search procedure is negligible if the factorization of G into low-rank matrices (i.e., $G = B_W E^{-1} B_W^T$ with $B_W \in \mathbb{R}^{n\times p}$, $p \ll n$) is exploited; see (Benner *et al.*, 2000a) for details.

The following theorem summarizes the convergence properties of Algorithm 1. Here we need the notion of a stabilizable pair of matrices: $(F,G) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m}$ is *stabilizable* if there exists $Z \in \mathbb{R}^{m\times n}$ such that $F + GZ$ is stable.

Theorem 2. *If $G \geq 0$, (F,G) is stabilizable, a unique stabilizing solution X_* of (24) exists, X_0 is stabilizing, and on the assumption that the local minimizer $t_j \in [0,2]$ of f_j in (26) satisfies $t_j \geq t_L > 0$ for all $j \in \mathbb{N}_0$, where $t_L > 0$ is a constant tolerance threshold, the sequence of approximate solutions X_j produced by Algorithm 1 has the following properties:*

(i) X_j *is stabilizing for all* $j \in \mathbb{N}_0$.

(ii) $\|\mathcal{R}(X_{j+1})\|_F \leq \|\mathcal{R}(X_j)\|_F$ *and equality holds if and only if* $\mathcal{R}(X_j) = 0$.

(iii) $\lim_{j\to\infty} \mathcal{R}(X_j) = 0$.

(iv) $\lim_{j\to\infty} X_j = X_*$.

(v) In a neighborhood of X_*, *the convergence is quadratic.*

(vi) $\lim_{j\to\infty} t_j = 1$.

For Newton's method, i.e., $t_j \equiv 1$, this theorem is proved in (Lancaster and Rodman, 1995). A proof of the above theorem using the stronger assumption of controllability can be found in (Benner and Byers, 1998) for (23), and (Benner, 1997) in (24). The controllability assumption was then weakened to stabilizability for (23) in (Guo and Laub, 2000). The argument used there extends immediately to the case of the ARE (24).

Bounding the step size t_j from below by a constant t_L is a common feature of line search procedures. This can be motivated based on the *Armijo rule*; see, e.g., (Dennis and Schnabel, 1983, Ch.6). We will elaborate on this, following the presentation used in (Dennis and Schnabel, 1983, Ch.6), as this aspect is important for a successful implementation of the method. Let $f : \mathbb{R}^\ell \to \mathbb{R}$ be twice continuously differentiable. If $x_j \in \mathbb{R}^\ell$ is an approximation to a local minimizer of $f(x)$ and p_j is a search direction for f from x_j, then a step size parameter t_j is accepted if it satisfies the sufficient decrease criterion given by the Armijo rule

$$f(x_j + t_j p_j) \leq f(x_j) + \alpha t_j \big(\nabla f(x_j)\big)^T p_j, \tag{27}$$

where $\alpha \in (0, 1)$. In order to translate (27) to the situation here, we use the usual embedding of $\mathbb{R}^{n\times n}$ into $\mathbb{R}^{n^2}$ given by the map $\mathrm{vec} : \mathbb{R}^{n\times n} \longrightarrow \mathbb{R}^{n^2}$,

$$\mathrm{vec}\,(X) := [x_{11}, x_{21}, \ldots, x_{n1}, x_{12}, \ldots, x_{n2}, \ldots, x_{1n}, \ldots, x_{nn}]^T,$$

which is obtained by consecutively stacking the columns of a matrix into a vector (see, e.g., (Lancaster and Tismenetsky, 1985, Ch.12)). Setting $x := \mathrm{vec}\,(X)$, $r(x) := \mathrm{vec}\,(\mathcal{R}\,(X))$ and

$$f(x) = \frac{1}{2} r(x)^T r(x) = \frac{1}{2}\mathrm{trace}\,\Big(\mathcal{R}\,(X)^T\,\mathcal{R}\,(X)\Big) = \frac{1}{2}\|\mathcal{R}\,(X)\,\|_F^2,$$

it can be shown that $\nabla f(\mathrm{vec}\,(X_j)) = \Gamma_j^T \mathrm{vec}\,(\mathcal{R}\,(X_j))$. Here $\Gamma_j = (F + GX_j)^T \otimes I_n + I_n \otimes (F + GX_j)^T$ denotes the matrix representation of the Lyapunov operator, mapping a symmetric matrix $Z \in \mathbb{R}^{n\times n}$ to $(F + GX_j)^T Z + Z(F + GX_j)$. As all X_j are stabilizing, it follows that all Γ_j are nonsingular (see, e.g., (Lancaster and Tismenetsky, 1985, Ch.13)). Hence

$$p_j := \mathrm{vec}\,(N_j) = -\Gamma_j^{-1}\mathrm{vec}\,(\mathcal{R}\,(X_j)) = -\Gamma_j^{-1} r(x_j)$$

is well defined and we obtain

$$\big(\nabla f(x_j)\big)^T p_j = -r(x_j)^T \Gamma_j \Gamma_j^{-1} r(x_j) = -\|\mathcal{R}\,(X_j)\,\|_F^2 < 0. \tag{28}$$

From (28) it can be deduced (Dennis and Schnabel, 1983) that the Newton direction provides a *descent direction* for $f(\mathrm{vec}\,(X))$ from X_j. Moreover, the Armijo rule (27) translates to

$$\|\mathcal{R}\,(X_j + t_j N_j)\,\|_F \leq \sqrt{1 - 2\alpha t_j}\,\|\mathcal{R}\,(X_j)\,\|_F. \tag{29}$$

If we choose a constant $\alpha \in (0, 1/4)$ and if the step sizes t_j are chosen from $[\,t_L,\, 2\,]$ for a given (small) constant $t_L > 0$, then

$$0 < 1 - 2\alpha t_j \leq 1 - 2\alpha t_L := \gamma^2 < 1, \tag{30}$$

for all $j = 0, 1, 2, \ldots$. Hence,

$$\|\mathcal{R}(X_j)\|_F \leq \gamma^j \|\mathcal{R}(X_0)\|_F. \tag{31}$$

As $\gamma < 1$, $\lim_{j\to\infty} \|\mathcal{R}(X_j)\|_F = 0$. Therefore, we obtain global convergence of $\mathcal{R}(X_j)$ to zero without using the control-theoretic assumptions of Theorem 2. But note that this does not necessarily imply $X_j \to X_*$. This is where the additional assumptions are needed in the convergence proofs in (Benner, 1997; Benner and Byers, 1998; Guo and Laub, 2000).

Besides ensuring theoretical convergence, requiring (29) is also reasonable as otherwise a stagnation of the line search method is possible. The same arguments can be used to explain why the assumption $t_j \geq t_L$ is not only a technical assumption but should be a requirement: choosing t_j too small will result in almost no progress towards the attractor of the iteration.

The assumptions of Theorem 2 and (29) are ensured by employing the following considerations:

- Set a lower bound t_L for the step size parameters t_j. (Numerical experiments indicate that $t_L = 10^{-4}$ is a reasonable choice.) Select the parameter $\alpha \in (\,0,\, 1/4\,)$ for the Armijo rule. In concurrent optimization literature (Dennis and Schnabel, 1983), $\alpha = 10^{-4}$ is proposed. Together with t_L as above, γ^2 in (30) is $\approx 1 - 10^{-8}$, which may yield very slow convergence—something we try to avoid by using line searches. Therefore we propose here a much larger value for α. In our experiments, $\alpha = 0.2$ worked very well.

- In each step, we compute the local minimizer $t_j \in [\,0,\, 2\,]$ and then set $t_j = \max\{t_j, t_L\}$ unless $t_j = 0$. (If $t_j = 0$, X_j is a solution of (24) and we can stop the iteration.)

- The condition (29) can be checked by evaluating $f_j(t)$ using (26) rather than by forming the residual matrix explicitly. If (29) is not satisfied in some step, there are several options:

  - Find another line search parameter t_j by a backtracking strategy as described in (Dennis and Schnabel, 1983);
  - decrease α or t_L;
  - perform a standard Newton step with $t_j = 1$.

  The second option usually results in a stagnation of the iteration as no significant progress will be achieved. The third option can be considered as restarting the Newton iteration from a new initial guess. Note that the convergence theory given in Theorems 2 ensures that the new "starting guess" is again stabilizing. One should limit the number of allowed restarts and run the Newton iteration without line search if this number is exceeded. This guarantees global convergence for the overall process.

In addition to the above, we also employ a criterion to overcome possible stagnation. This criterion is defined as follows:

$$\|\mathcal{R}(X_j + t_j N_j)\|_F < \mathrm{tol}_S \|\mathcal{R}(X_{j-j_S})\|_F. \tag{32}$$

For this, we only need to keep the norms of the last j_S residuals in memory. In case of stagnation we again "restart" the method using a Newton step. From numerical experience, we suggest to use $j_S = 2,\ \mathrm{tol}_S = 0.9$.

Here we apply the suggested modified Newton's method to (5) using (25). It can be proved that $A - B_W E^{-1} C$ is stable (Anderson, 1967a; 1967b; Green, 1988a; Varga and Fasol, 1993) and hence F_0 is stable. Therefore, we can start the Newton iteration with $X_0 = 0$ such that the usually difficult problem of finding a stabilizing starting guess is circumvented.

4. Parallel Implementation and Numerical Examples

4.1. Implementation Details

The numerical algorithms that we have described in the previous two sections are all composed of basic matrix computations such as linear systems, matrix products, QR factorizations (with column pivoting), etc. All these operations can be efficiently implemented on modern serial and parallel computers. Although one could develop his/her own parallel routines for this purpose, nowadays there exist libraries of parallel kernels for distributed memory computers (Blackford *et al.*, 1997). The use of these libraries enhances the reliability and improves the portability of the model reduction routines. The performance will depend on the efficiency of the underlying serial and parallel libraries and the communication routines.

Here we will employ the ScaLAPACK parallel library (Blackford *et al.*, 1997). This is a public domain library that implements parallel versions of many of the kernels in LAPACK (Anderson *et al.*, 1995), using the message-passing paradigm. The ScaLAPACK is based on PB-BLAS (the parallel version of the serial BLAS) for computation and BLACS for communication. The BLACS can be ported to any (serial and) parallel architecture with an implementation of the MPI (our case) or the PVM communication libraries (Geist *et al.*, 1994; Gropp *et al.*, 1994).

In ScaLAPACK the computations are performed on a logical grid of $n_p = p_r \times p_c$ processes. The processes are mapped onto the physical processors, depending on the available number of these. All data (matrices) have to be distributed among the process grid prior to the invocation of a ScaLAPACK routine. It is the user's responsibility to perform this data distribution. Specifically, in ScaLAPACK the matrices are partitioned into $nb \times nb$ square blocks and these blocks are distributed (and stored) among the processes in column-major order. See (Blackford *et al.*, 1997) for details.

Employing ScaLAPACK, we have implemented the following BST model reduction algorithms for LTI systems as Fortran 77 subroutines:

- `PDGESTSR`: the BST SR method for model reduction;
- `PDGESTBF`: the BST BFSR method for model reduction.

We compare these parallel routines with the analogous serial algorithms in SLICOT[2]. This library includes the following Fortran 77 routine:

- `AB09HD`: the BST SR or BFSR method;

All the experimental results described in the following two subsections were obtained on a Beowulf cluster. Each node consists of an Intel Pentium-II processor at 300 MHz and 128 MBytes of RAM. All computations are performed in IEEE double-precision floating-point arithmetic (i.e., the machine epsilon was $\varepsilon \approx 2.2204 \times 10^{-16}$). We employ a BLAS library, specially tuned for the Pentium-II processor, which achieves around 180 Mflops (millions of flops per second) for the matrix product (routine `DGEMM`). The nodes are connected by a *Myrinet* switch, and the communication library BLACS employs a tuned implementation of MPI. The performance of the interconnection network for MPI was measured by a simple loopback message transfer and offered a latency of 33 μsec and a bandwidth around 200 Mbit/sec. We made extensive use of the LAPACK, PB-BLAS, and ScaLAPACK libraries.

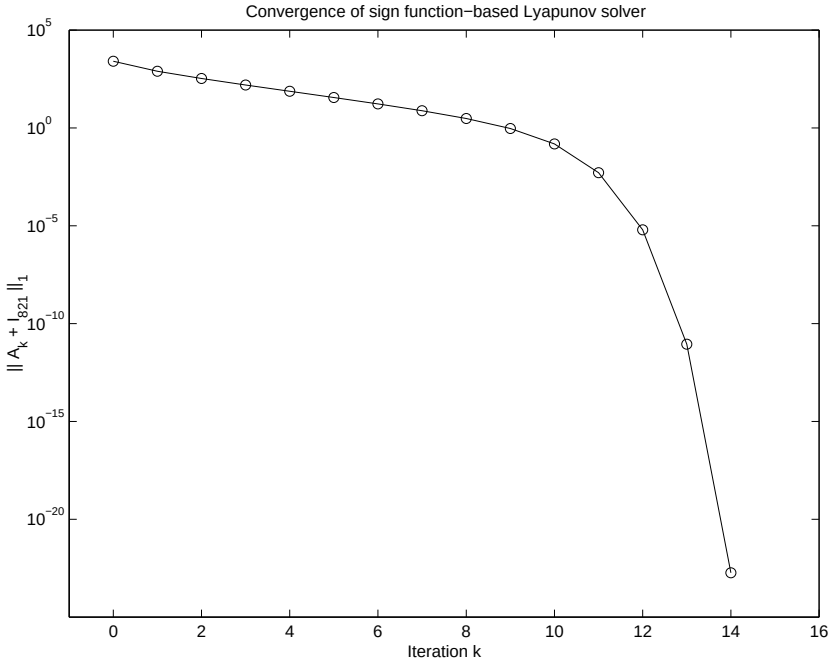
4.2. Numerical Accuracy

In order to demonstrate the model reduction performance of our parallel implementation of the BST method based on the Lyapunov and ARE solvers suggested in Section 3, we apply the algorithm to an example from (Penzl, 1999).

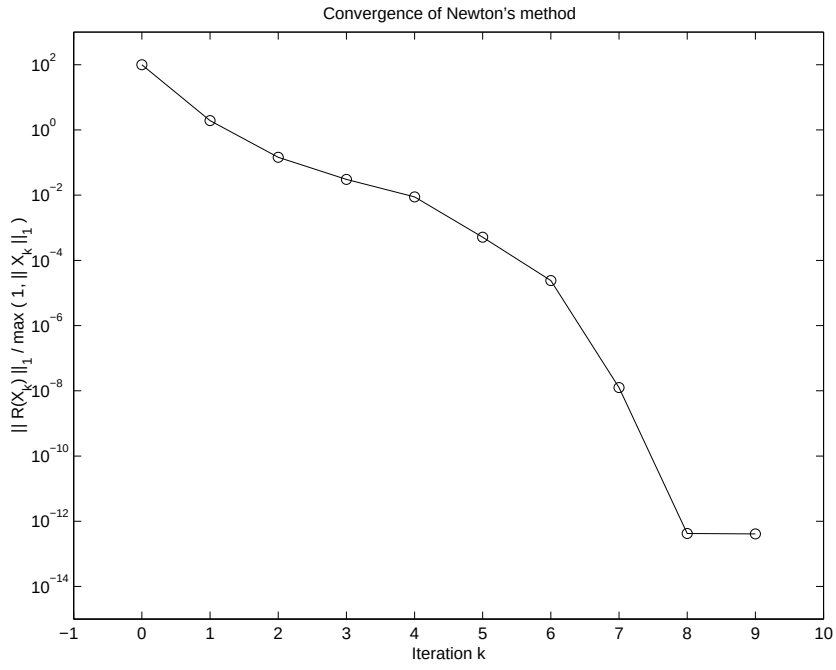
The LTI system comes from a mathematical model for optimal cooling of steel profiles. The process is modeled by a boundary control problem for a linearized 2-dimensional heat equation. A finite element discretization of the spatial variables results in a square system with 6 inputs and outputs. The state-space dimension is $n = 821$ for a coarse mesh and $n = 3113$ for an adaptively refined mesh. As $D = 0$ in this example, we use an ϵ-regularization and set $D = 10^{-1} I_6$. We compute a reduced-order model of size $r = 40$. Fig. 1(a) shows the convergence behavior of the factored Lyapunov solver proposed in Section 3.1 applied to (4) in terms of $\|A_k + I_n\|$. Fig. 1(b) displays the normalized residuals during the Newton iteration when solving (5). For all Lyapunov equations involved, i.e., (4), (9), and the Lyapunov equations to be solved in each Newton iteration, the sign function-based solvers with determinantal scaling needed between 12 and 17 iterations to converge.

Our parallel routines computed full-rank factors $\hat{S}$ and $\hat{R}$ of the Gramians of rank 163 and 199, respectively. The resulting values σ_j from (17) are plotted in Fig. 2. The plot shows a fast decay of the singular values, in particular taking into

[2]Available via anonymous ftp at `ftp://www.esat.kuleuven.ac.be/pub/WGS/SLICOT`



(a)



(b)

Fig. 1. Convergence of iterative methods when solving (4) (a) and (5) (b).

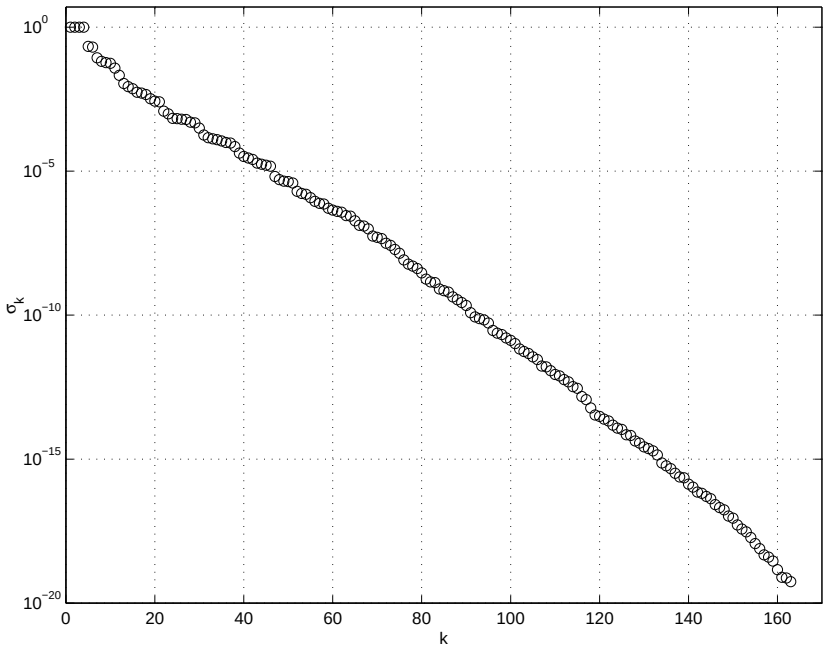


Fig. 2. Singular values of $\hat{S}\hat{R}^T$.

account the fact that 658 numerically zero singular values are not shown. The error bound (8) yields

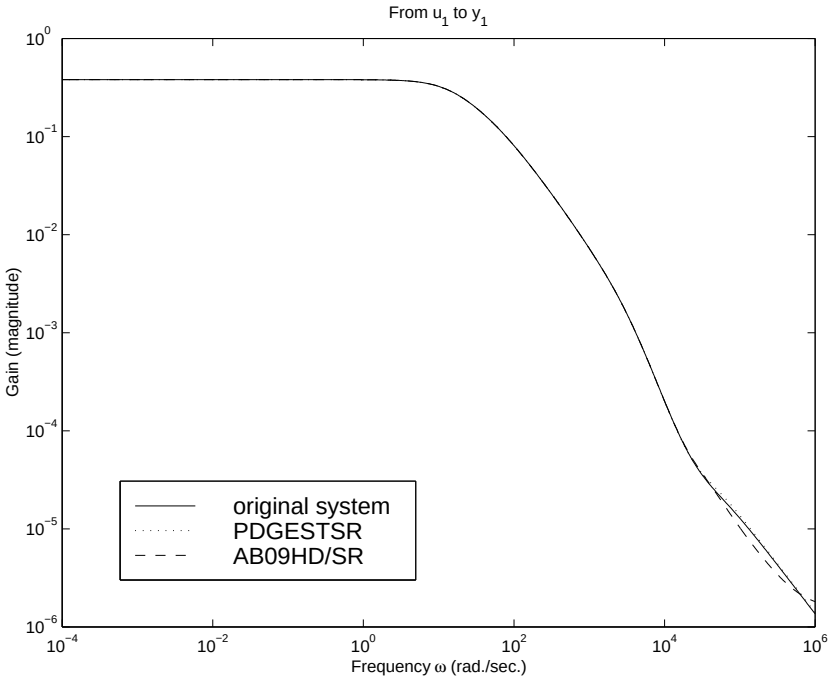
$$\|G - G_{40}\|_\infty \leq 3.1495 \cdot 10^{-4} \cdot \|G\|_\infty. \tag{33}$$

Note that $G - G_{40}$ is equal for the original system or the ϵ-regularized system as $D = D_{40}$. In order for the above bound to hold, we need to use the ϵ-regularized system in the right-hand side expression as the bound (8) only holds for full-rank D. Figure 3 shows the *Bode (magnitude) plots* for four of the 36 input-output channels, comparing the input-output mapping (frequency response) of the original model and the reduced-order model computed with `PDGESTSR`. The differences in the performance of the original system and the reduced-order model are only visible for the very small magnitudes at high frequencies, i.e., at the noise level. This demonstrates that the reduced-order model satisfactorily approximates the original system.

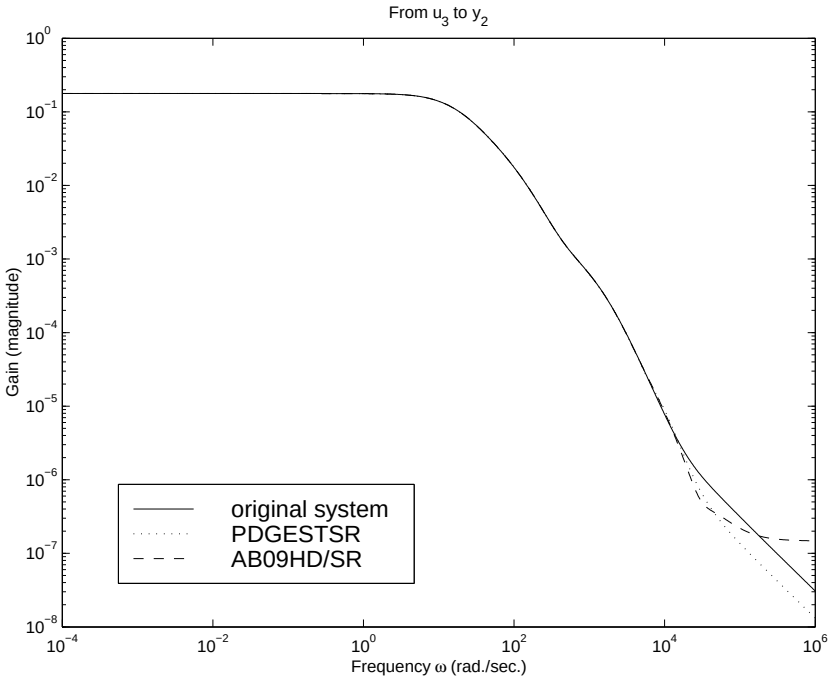
This statement is even more supported by checking the singular values of the error system $H := G - G_{40}$. In Fig. 4, the maximum singular values of the error system are plotted. That is, the figure shows the absolute error

$$\|G(\jmath\omega) - G_{40}(\jmath\omega)\|_2 = \sigma_{\max}\big(H(\jmath\omega)\big)$$

for the reduced-order models computed by our routine `PDGESTSR` and the SLICOT routine `AB09HD` (using the SR version). From the figure it is obvious that both reduced-order models satisfy the theoretical absolute error bound (33). For small frequencies,

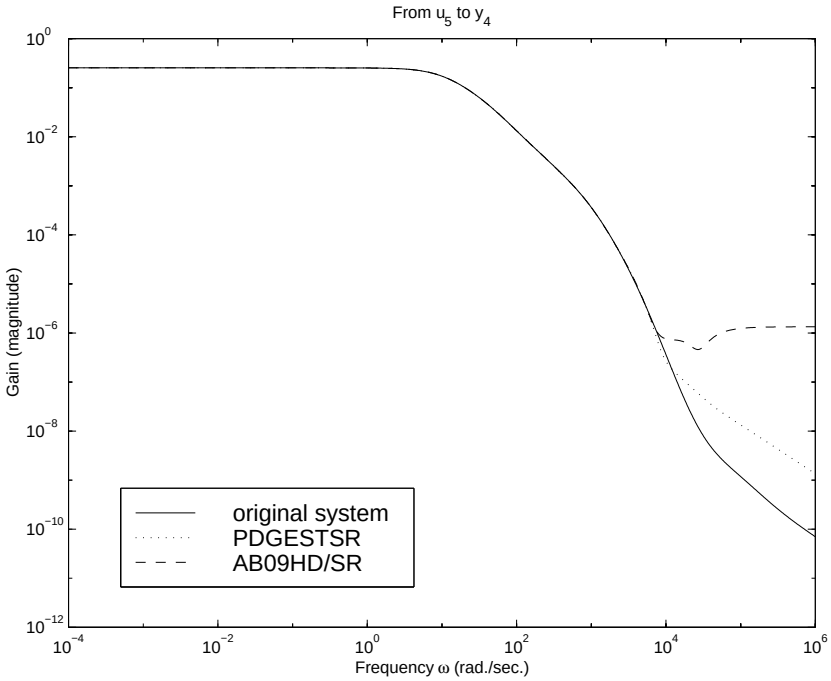


(a)

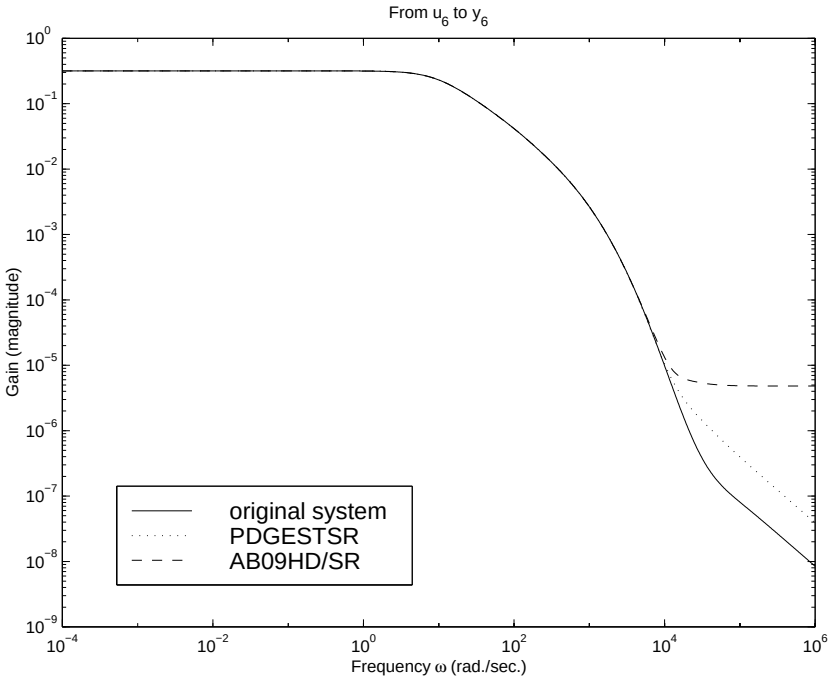


(b)

Fig.3. Bode magnitude plots for original and reduced-order models.



(c)



(d)

Fig. 3. Bode magnitude plots for original and reduced-order models (continued).

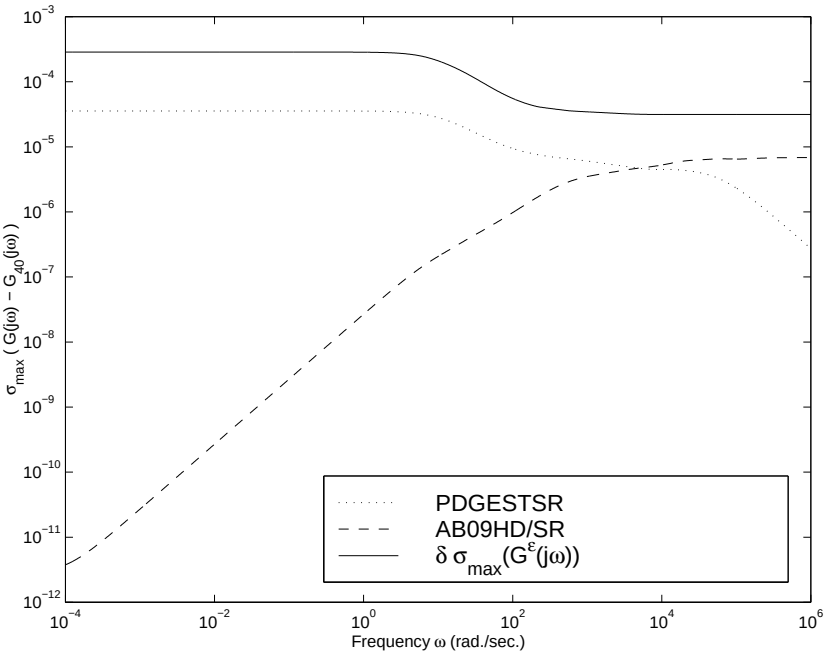


Fig. 4. Singular value plot for error systems.

SLICOT produces a very small error while for increasing frequencies, the model computed by `PDGESTSR` appears to approximate the system slightly better.

For this example, we need to run our algorithm on four processors to beat the execution time of the SLICOT routine `AB09HD`. Roughly speaking, the timings are 1950 seconds for `AB09HD`, and 2450 and 1630 seconds for `PDGESTSR` on two and four processors, respectively. This is due to the fact that we needed 10 Newton iterations to solve the ARE (5). In flops, this is roughly equivalent to twice the time needed to solve an ARE using the Schur vector method employed in `AB09HD`. But note that using the full-rank factors of the Gramians as in our approach, the SVD (17) needs less than 1 second using 2 or 4 processors, while computing the SVD (10) employing the full Cholesky factors takes 54 seconds on one processor.

The situation becomes a bit different for the model of order $n = 3113$ resulting from a finer discretization. For this system we are able to compute a reduced-order model of size $r = 137$ on 16 processors in 6 hours and 32 minutes. The iteration numbers for Newton's method to solve the ARE and for solving the Lyapunov equations involved are similar to those for $n = 821$, i.e., 10 Newton iterations and 14–19 sign function iterations for solving the Lyapunov equations.

Note that for this problem size there is no sufficient memory on one processor to compute a reduced-order model using `AB09HD`. The ranks of the full-rank factors computed with `PDGESTSR` are 179 and 204—approximately the same as for the case of $n = 821$. Hence the execution time for the SVD (17) is approximately the same

as for $n = 821$. But extrapolating the execution time for the SVD (10) from 821 to 3113 results in approximately 50 minutes while the total extrapolated execution time for `AB09HD` would be around 29.5 hours. Though this still shows no perfect speed-up (only a factor 4.5 using 16 processors), parallel computing enabled us to solve this problem, which was not possible using just one processor.

4.3. Parallel Performance

In this subsection we analyze the performance of the algorithms on a parallel distributed Beowulf cluster as described in Subsection 4.1. For the parallel numerical experiments used to measure the performance, we generate square LTI systems with prescribed poles, i.e., the eigenvalues of A were fixed in an interval $[-1000, -1]$. This resembles the eigenvalue distribution of a (scaled) discrete one-dimensional Laplace operator. B and C were chosen randomly with $m = p = 10$ and we set $D = I_{10}$.

Our first experiment evaluates the reduction in the execution time achieved by the parallel BST algorithms. For this purpose we compare the execution time of the SLICOT routine `AB09HD/SR` (Varga, 1999) with that of our parallel routine `PDGESTSR`. (The results for the BFSR algorithms showed no significant difference with respect to those reported here.) In the experiment, n is set to 1000, and we choose $m = p = 10$. Figure 5 shows the execution times of the serial algorithm from SLICOT (the execution time on one processor) and those of the parallel algorithm for several numbers of nodes, n_p. The figure shows a significant reduction in execution time

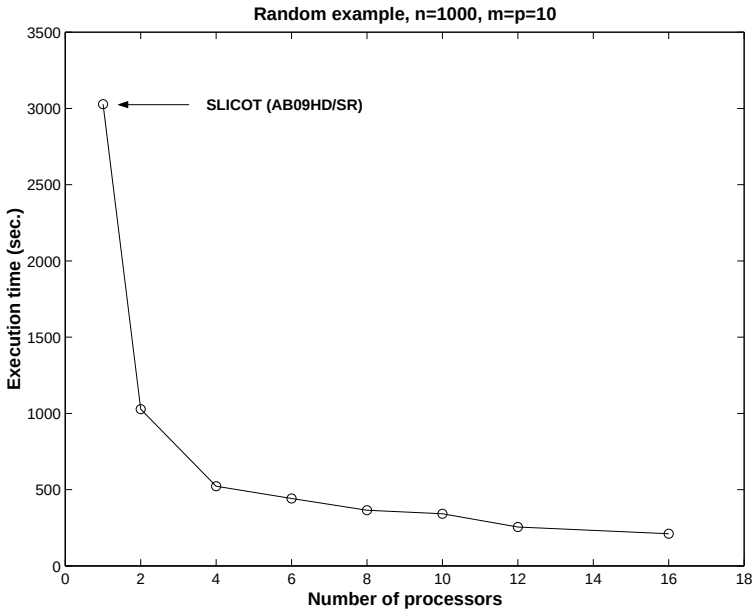


Fig. 5. Execution time vs. number of nodes of the serial and parallel BST SR algorithms.

from one to two processors, which is partly due to the higher efficiency of our BST algorithm for this example and partly to the use of two processors instead of one. The speed-up from two to four processors is nearly perfect while for larger numbers of processors the parallel efficiency decreases. This is due to the fact that the problem size is too small to fully exploit parallelism on more than four processors. It should also be noted that, for this problem class, speed may not be the only performance index. On just one of the nodes of our Beowulf cluster, problems of size $n > 2000$ (e.g., the $n = 3113$ example in the last section) cannot be solved due to memory limitations. This becomes possible only by connecting them to a cluster such that the data can be distributed to the nodes. In that way, much larger problems can be solved.

We next report the scalability of the parallel algorithms. The scalability evaluates whether a larger problem can be solved by increasing proportionally the number of nodes of the parallel system. In the experiment we fix the problem size per node at $n/\sqrt{n_p} = 800,\ m/\sqrt{n_p} = 400 = q_s/\sqrt{n_p},\ p/\sqrt{n_p} = 400 = q_r/\sqrt{n_p}$. In Fig. 6 we report the Mflop ratio per node for the parallel algorithms `PDGESTSR` and `PDGESTBF`.

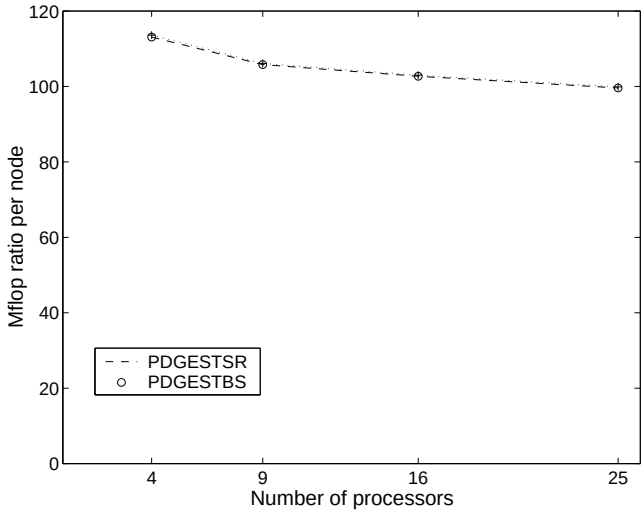


Fig. 6. Scalability of the parallel algorithms with $n/\sqrt{n_p} = 800$, $m/\sqrt{n_p} = 400 = q_s/\sqrt{n_p},\ p/\sqrt{n_p} = 400 = q_r/\sqrt{n_p}$.

The figure shows a high scalability of the algorithms since there is only a minor decrease in the Mflop ratio per node as the number of nodes is increased up to 25 (a problem of size n=4000).

5. Conclusions

We have described efficient and reliable numerical algorithms for the realization of model reduction methods based on the square-root version of balanced stochastic

truncation. Using the full-rank factors of the Gramians often enhances the efficiency and accuracy of these methods significantly. The proposed method employs efficient algorithms for solving stable Lyapunov equations based on the Newton iteration for the sign function method. Using this algorithm inside Newton's method for solving the algebraic Riccati equation which yields the observability Gramian in this context allows us to solve relatively large problems on parallel computers.

Implementations of the discussed methods are based on highly optimized software packages for numerical linear algebra on serial and parallel computers. Our experiments report similar numerical results for reliable serial model reduction algorithms from the SLICOT library and our model reduction approach, based on the sign function method. The results on a cluster of Intel Pentium-II nodes show the performance of our model reduction approach and the scalability of the parallel algorithms. Parallel computing thus allows us to use these methods for systems of state-space dimension up to order $\mathcal{O}(10^4)$.

References

Anderson B.D.O. (1967a): *An algebraic solution to the spectral factorization problem.* — IEEE Trans. Automat. Contr., Vol.AC–12, pp.410–414.

Anderson B.D.O. (1967b): *A system theory criterion for positive real matrices.* — SIAM J. Contr., Vol.5, No.2, pp.171–182.

Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Ostrouchov S. and Sorensen D. (1995): *LAPACK Users' Guide, 2nd Ed.* — Philadelphia, PA: SIAM.

Bai Z. and Demmel J. (1993): *Design of a parallel nonsymmetric eigenroutine toolbox, Part I*, Proc. 6-th SIAM Conf. *Parallel Processing for Scientific Computing*, pp.391–398. SIAM, Philadelphia, PA. See also: Tech. Rep. CSD–92–718, Computer Science Division, University of California, Berkeley, CA 94720.

Bai Z., Demmel J., Dongarra J., Petitet A., Robinson H. and Stanley K. (1997): *The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers.* — SIAM J. Sci. Comput., Vol.18, No.5, pp.1446–1461.

Bartels R.H. and Stewart G.W. (1972): *Solution of the matrix equation $AX + XB = C$: Algorithm 432.* — Comm. ACM, Vol.15, No.9, pp.820–826.

Benner P. (1997): *Numerical solution of special algebraic Riccati equations via an exact line search method.* — Proc. European Control Conf. *ECC 97*, Brussels, Belgium, Paper 786. (CD-ROM).

Benner P. and Byers R. (1998): *An exact line search method for solving generalized continuous-time algebraic Riccati equations.* — IEEE Trans. Automat. Contr., Vol.43, No.1, pp.101–107.

Benner P. and Quintana-Ortí E.S. (1999): *Solving stable generalized Lyapunov equations with the matrix sign function.* — Numer. Algorithms, Vol.20, No.1, pp.75–100.

Benner P., Claver J.M. and Quintana-Ortí E.S. (1999): *Parallel distributed solvers for large stable generalized Lyapunov equations.* — Parall. Process. Lett., Vol.9, No.1, pp.147–158.

Benner P., Byers R., Quintana-Ortí E.S. and Quintana-Ortí G. (2000a): *Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search.* — Parallel Comput., Vol.26, No.10, pp.1345–1368.

Benner P., Quintana-Ortí E.S. and Quintana-Ortí G. (2000b): *Balanced truncation model reduction of large-scale dense systems on parallel computers.* — Math. Comp. Model. Dyn. Syst., Vol.6, No.4, pp.383–405.

Bischof C.H. (1990): *Incremental condition estimation.* — SIAM J. Matrix Anal. Appl., Vol.11, No.2, pp.312–322.

Blackford L.S., Choi J., Cleary A., D'Azevedo E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D. and Whaley R.C. (1997): *ScaLAPACK Users' Guide.* — Philadelphia, PA: SIAM.

Byers R. (1987): *Solving the algebraic Riccati equation with the matrix sign function.* — Lin. Alg. Appl., Vol.85, pp.267–279.

Dennis J. and Schnabel R.B. (1983): *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* — Englewood Cliffs: Prentice Hall.

Desai U.B. and Pal D. (1984): *A transformation approach to stochastic model reduction.* — IEEE Trans. Automat. Contr., Vol.AC–29, No.12, pp.1097–1100.

Dongarra J.J., Sameh A. and Sorensen D. (1986): *Implementation of some concurrent algorithms for matrix factorization.* — Parall. Comput., Vol.3, pp.25–34.

Fernando K.V. and Hammarling S.J. (1988): *A product induced singular value decmoposition for two matrices and balanced realization*, In: Linear Algebra in Signals, Systems and Control (B.N. Datta, C.R. Johnson, M.A. Kaashoek, R. Plemmons and E. Sontag, Eds). — Philadelphia, PA: SIAM, pp.128–140.

Gardiner J.D. and Laub A.J. (1991): *Parallel algorithms for algebraic Riccati equations.* — Int. J. Contr., Vol.54, No.6, pp.1317–1333.

Geist A., Beguelin A., Dongarra J., Jiang W., Manchek B. and Sunderam V. (1994): *PVM: Parallel Virtual Machine—A Users Guide and Tutorial for Network Parallel Computing.* — Cambridge, MA: MIT Press.

Glover K. (1986): *Multiplicative approximation of linear multivariable systems with L_∞ error bounds.* — Proc. American Control Conf., Seattle, WA, pp.1705–1709.

Golub G.H. and Van Loan C.F. (1996): *Matrix Computations, 3rd Ed.* — Baltimore: Johns Hopkins University Press.

Green M. (1988a): *Balanced stochastic realization.* — Lin. Alg. Appl., Vol.98, pp.211–247.

Green M. (1988b): *A relative error bound for balanced stochastic truncation.* — IEEE Trans. Automat. Contr., Vol.AC–33, No.10, pp.961–965.

Gropp W., Lusk E. and Skjellum A. (1994): *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* — Cambridge, MA: MIT Press.

Guo C.-H. and Laub A.J. (2000): *On a Newton-like method for solving algebraic Riccati equations.* — SIAM J. Matrix Anal. Appl., Vol.21, No.2, pp.694–698.

Hammarling S.J. (1982): *Numerical solution of the stable, non-negative definite Lyapunov equation.* — IMA J. Numer. Anal., Vol.2, pp.303–323.

Heath M.T., Laub A.J., Paige C.C. and Ward R.C. (1987): *Computing the SVD of a product of two matrices.* — SIAM J. Sci. Statist. Comput., Vol.7, pp.1147–1159.

Kenney C. and Laub A.J. (1995): *The matrix sign function.* — IEEE Trans. Automat. Contr., Vol.40, No.8, pp.1330–1348.

Kleinman D.L. (1968): *On an iterative technique for Riccati equation computations.* — IEEE Trans. Automat. Contr., Vol.AC–13, No.2, pp.114–115.

Lancaster P. and Rodman L. (1995): *The Algebraic Riccati Equation.* — Oxford: Oxford University Press.

Lancaster P. and Tismenetsky M. (1985): *The Theory of Matrices, 2nd Ed.* — Orlando: Academic Press.

Larin V.B. and Aliev F.A. (1993): *Construction of square root factor for solution of the Lyapunov matrix equation.* — Syst. Contr. Lett., Vol.20, No.2, pp.109–112.

Laub A.J., Heath M.T., Paige C.C. and Ward R.C. (1987): *Computation of system balancing transformations and other application of simultaneous diagonalization algorithms.* — IEEE Trans. Automat. Contr., Vol.34, pp.115–122.

Liu Y. and Anderson B.D.O. (1986): *Controller reduction via stable factorization and balancing.* — Int. J. Contr., Vol.44, pp.507–531.

Moore B.C. (1981): *Principal component analysis in linear systems: Controllability, observability, and model reduction.* — IEEE Trans. Automat. Contr., Vol.AC–26, No.2, pp.17–32.

Penzl T. (1999): *Algorithms for model reduction of large dynamical systems.* — Tech. Rep. SFB393/99–40, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG. Available from `http://www.tu-chemnitz.de/sfb393/sfb99pr.html`.

Penzl T. (2000): *Eigenvalue decay bounds for solutions of Lyapunov equations: the symmetric case.* — Syst. Contr. Lett., Vol.40, pp.139–144.

Roberts J.D. (1980): *Linear model reduction and solution of the algebraic Riccati equation by use of the sign function.* — Int. J. Contr., Vol.32, pp.677–687. (Reprint of Technical Report No.TR–13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

Safonov M.G. and Chiang R.Y. (1988): *Model reduction for robust control: A Schur relative error method.* — Int. J. Adapt. Cont. Sign. Process., Vol.2, pp.259–272.

Tombs M.S. and Postlethwaite I. (1987): *Truncated balanced realization of a stable non-minimal state-space system.* — Int. J. Contr., Vol.46, No.4, pp.1319–1330.

Varga A. (1995): *On computing high accuracy solutions of a class of Riccati equations.* — Contr. Theory Adv. Technol., Vol.10, No.4, pp.2005–2016.

Varga A. (1999): *Task II.B.1— selection of software for controller reduction.* — SLICOT Working Note 1999–18, The Working Group on Software (WGS). Available from `http://www.win.tue.nl/niconet/NIC2/reports.html`.

Varga A. and Fasol K.H. (1993): *A new square–root balancing–free stochastic truncation model reduction algorithm.* — Prepr. 12-th IFAC World Congress, Vol.7, pp.153–156, Sydney, Australia.