Sergio Barrachina   Peter Benner   Enrique S. Quintana-Ortí

Solving Large-Scale Generalized Algebraic Bernoulli Equations via the Matrix Sign Function

Preprint SFB393/05-15

Abstract

We investigate the solution of large-scale generalized algebraic Bernoulli equations as those arising in control and systems theory in the context of stabilization of linear dynamical systems, coprime factorization of rational matrix-valued functions, and model reduction.

The algorithms we propose, based on a generalization of the Newton iteration for the matrix sign function, are easy to parallelize, yielding an efficient numerical tool to solve large-scale problems. Both the accuracy and the parallel performance of our implementations on a cluster of Intel Xeon processors are reported.

Key words: Bernoulli equation, linear and nonlinear matrix equations, matrix sign function, control and systems theory, parallel computers.

Preprintreihe des Chemnitzer SFB 393

ISSN 1619-7178 (Print)   ISSN 1619-7186 (Internet)

SFB393/05-15   **December 2005**

Contents

Authors' addresses

Sergio Barrachina
Enrique S. Quintana-Ortí

Depto. de Ingeniería y Ciencia
de Computadores, Universidad Jaume I,
12.071–Castellón, Spain

barrachi@icc.uji.es
quintana@icc.uji.es

Peter Benner

Fakultät für Mathematik,
Technische Universität Chemnitz,
D-09107 Chemnitz, Germany

benner@mathematik.tu-chemnitz.de

Sergio Barrachina and E.S. Quintana-Ortí were supported by the CICYT project No. TIN2005-09037-C02-02 and FEDER, and project No. P1B-2004-6 of the *Fundació Caixa-Castelló/Bancaixa and UJI.* P. Benner was supported by the DFG Sonderforschungsbereich 393 *Parallele Numerische Simulation für Physik und Kontinuumsmechanik* at TU Chemnitz.

1 Introduction

We consider the *generalized algebraic Bernoulli equation (GABE)*

$$A^T XE + E^T XA - E^T XGXE = 0, \tag{1}$$

where $E, A, G \in \mathbb{R}^{n\times n}$, $G = G^T$, and $X \in \mathbb{R}^{n\times n}$ is the sought-after solution. Equation (1) is a homogeneous *generalized algebraic Riccati equation* (GARE), i.e., a GARE with a zero constant term. Also, in correspondence with *Bernoulli's differential equation*

$$\dot{y}(t) = p(t)y(t) + q(t)y(t)^k, \quad k \neq 0, 1,$$

the name *generalized algebraic Bernoulli equation* can represent any equation of the form

$$\mathcal{L}(X) + A_0 X \left(\prod_{j=1}^{k-1} A_j X \right) A_k = 0,$$

where $\mathcal{L}(X)$ is a linear operator and $A_j \in \mathbb{R}^{n\times n}$ for $j = 0, 1, \ldots, k$. Thus, the GABE in (1) is actually a special Bernoulli equation (with $k = 2$, $A_0 = E^T$, $A_1 = G$, $A_2 = E$) as well as a special GARE.

The numerical solution of the GABE is required in several coprime factorization problems for rational transfer functions used in robust control (see, e.g., [1] or [2, Section 13.7]) and also in recent methods for model reduction of unstable linear dynamical systems [3]. Large-scale linear dynamical systems of this type arise, e.g., when modeling RLC circuits and VLSI devices [4]. Here, the attribute "large" means that the dimension of the GABE, n, can range from 10^3 to 10^5. Numerically reliable methods as those described below require $\mathcal{O}(n^3)$ flops (floating-point arithmetic operations) and storage for $\mathcal{O}(n^2)$ real numbers. Thus, the solution of a GABE of dimension n in the thousands (or larger) using these methods greatly benefits from the use of parallel computing techniques.

By considering the GABE as a degenerate case of the GARE,

$$A^T XE + E^T XA - E^T XGXE + Q = 0,$$

well-known methods for solving this latter equation can also be applied to the GABE. Thus, we can obtain a solution of the GABE from any n-dimensional deflating subspace of the $2n \times 2n$ matrix pencil

$$H - \lambda K := \begin{bmatrix} A & G \\ 0 & -A^T \end{bmatrix} - \lambda \begin{bmatrix} E & 0 \\ 0 & E^T \end{bmatrix} \tag{2}$$

for which a basis, spanned by the columns of $\begin{bmatrix} U^T & V^T \end{bmatrix}^T$, $U, V \in \mathbb{R}^{n\times n}$, exists such that U is invertible. Then, a solution of the GABE can be computed as $X_* := -VU^{-1}E^{-1}$ [12] if E is nonsingular.

We can obtain the appropriate subspace of (2) by means of the QZ algorithm for the generalized (real) Schur form [6] or any of its structure-preserving variants [7], followed by a procedure to reorder the eigenvalues of the matrix pencil conveniently. However, this approach consists of fine-grain operations which are difficult to parallelize efficiently on distributed-memory architectures [8]. As a proof, there exists currently no parallel implementation of the QZ algorithm for distributed-memory platforms. Moreover, in some applications, the matrix G in the quadratic term of (1) is given in factored form (see below). Currently, there is no deflating subspace-based method that can exploit this low-rank structure.

In this paper we extend previous results in [9] for the standard case ($E = I_n$, the identity matrix of order n) of the ABE to the generalized equation. In particular, we investigate iterative solvers based on a generalization of the Newton iteration for the matrix sign function, which are specially appealing in that they can be easily parallelized and deliver notable performance on parallel distributed-memory architectures. We also show that an initial transformation of the GABE yields a second iterative scheme with a reduced computational cost. Moreover, for those applications in which the GABE appears in the *factored form*

$$A^T XE + E^T XA - E^T XBB^T XE = 0, \tag{3}$$

with $B \in \mathbb{R}^{n\times m}$ and $m \ll n$, we introduce an algorithm based on the sign function with the potential to deliver important savings during the iteration.

The paper is structured as follows. In Section 2 we briefly provide the theoretical background on the existence of the solutions to the GABE. In Section 3 we present an iterative scheme for solving the GABE based on the matrix sign function. The equation is transformed in Section 4 yielding an iterative scheme with a lower computational cost, and modified in Section 5 in order to deal with the factored form of the equation and further reduce the cost. Experiments reporting the numerical accuracy and the efficiency of a parallel implementation of the new method on a cluster of Intel Xeon processors are given in Section 6. The final section summarizes a few concluding remarks.

Throughout the paper we will use $\Lambda(M, N)$ to refer to the spectrum (set of eigenvalues) of a matrix pencil $M - \lambda N$, and $\mathbb{C}^-$, $\mathbb{C}^+$ to denote, respectively, the open left and right half planes. Also, I_p will stand for the identity matrix of order p, $\jmath := \sqrt{-1}$, and $\jmath\mathbb{R}$ will denote the imaginary axis. A symmetric positive (semi-)definite matrix M is indicated by writing $M > 0$ ($M \geq 0$).

2 Theoretical background

Let us first review a pair of classical definitions from control theory. Here and in the following we assume that E is nonsingular.

Definition 2.1 *Consider a generalized continuous linear time-invariant system defined in state-space form by the differential equation*

$$E\dot{x}(t) = Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \tag{4}$$

where $E, A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, *and* $x^0 \in \mathbb{R}^n$ *is the initial state of the system.*

a) The system (4) (or just the matrix pencil $A - \lambda E$) is said to be (asymptotically) stable/anti-stable *if* $\Lambda(A, E) \subset \mathbb{C}^- \;/\; \Lambda(A, E) \subset \mathbb{C}^+$.

b) The system (4) (or the matrix pair triple (E, A, B)) is said to be controllable *if for any initial and final states, x^0 and x^*, respectively, there exist a finite time t^* and an input $u(t)$, $0 \le t < t^*$, such that $x(t^*) = x^*$.*

Under the given assumptions, controllability is equivalent to the full rank of the *controllability matrix*

$$\mathcal{K} := \left[E^{-1}B, E^{-1}AE^{-1}B, \ldots, (E^{-1}A)^{n-1}E^{-1}B\right].$$

The solution of a GABE as in (1) —if it exists— is usually not unique. In most applications, one of the particular solutions, named in the following definition, is required.

Definition 2.2 *A solution of the GABE (1) is called* stabilizing *or* anti-stabilizing *if* $\Lambda((A - GX), E) \subset \mathbb{C}^-$ *or* $\Lambda((A - GX), E) \subset \mathbb{C}^+$, *respectively.*

Due to its nature as GARE, it is not surprising that the solution theory of the GABE can be derived from that of the GARE. This theory can be summarized in the following result.

Theorem 2.3 (Extremal GABE solutions) *Consider the GABE (1) with $G = BB^T \ge 0$ and (E, A, B) controllable. Then there exist symmetric solutions $X^+ \ge 0$, $X^- \le 0$ of (1) with $X^- \le X \le X^+$ for all solutions X of (1).*

Moreover, X^- is the unique solution satisfying $\Lambda((A - GX^-), E) \subset \mathbb{C}^+ \cup \jmath\mathbb{R}$ and X^+ is the unique solution satisfying $\Lambda((A - GX^+), E) \subset \mathbb{C}^- \cup \jmath\mathbb{R}$. If $\Lambda(A, E) \cap \jmath\mathbb{R} = \emptyset$, then X^- is the unique anti-stabilizing solution and X^+ is the unique stabilizing solution of the GABE.

Proof. Using the equivalence of (1) to

$$(AE^{-1})^T X + X(AE^{-1}) - XGX = 0,$$

the theorem is a trivial consequence of Theorem 7.5.1 and Section 8.3 of [10]. Theorem 7.5.1 in [10] is applicable as there always exists a positive semidefinite solution of (1), given by $X = 0$. $\square$

This result implies that for a stable matrix pencil $A - \lambda E$ all solutions of the GABE are negative semidefinite, while for an anti-stable pencil $A - \lambda E$ all solutions are positive semidefinite.

In the following we will concentrate on computing stabilizing solutions of (1). These are the solutions needed in stabilization of linear dynamical systems and model reduction of unstable systems [3], which are our primary areas of interest.

3 Solving the GABE with the matrix sign function

The matrix sign function has been shown to be a valuable tool for the numerical solution of linear matrix equations as, for example, Sylvester and Lyapunov equations, and quadratic matrix equations such as the GARE. We next illustrate how to adapt a generalization of the Newton iteration for the matrix sign function for the solution of the GARE.

Consider a matrix $Z \in \mathbb{R}^{\ell\times\ell}$ with no eigenvalues on the imaginary axis, and let

$$Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1}$$

be its Jordan decomposition. Here, the Jordan blocks in $J^- \in \mathbb{R}^{j\times j}$ and $J^+ \in \mathbb{R}^{(\ell-j)\times(\ell-j)}$ contain, respectively, those parts of the set of eigenvalues of Z in the open left and right half planes. The *matrix sign function* of Z is then defined as

$$\operatorname{sign}(Z) := S \begin{bmatrix} -I_j & 0 \\ 0 & I_{\ell-j} \end{bmatrix} S^{-1}.$$

By applying Newton's root-finding iteration to $Z^2 = I_l$, with the starting point chosen as Z, we obtain the Newton iteration for the matrix sign function:

$$Z_0 \leftarrow Z, \quad Z_{k+1} \leftarrow \tfrac{1}{2}(Z_k + Z_k^{-1}), \quad k = 0, 1, 2, \ldots. \tag{5}$$

The sequence $\{Z_k\}_{k=0}^{\infty}$ converges with an ultimately quadratic convergence rate and

$$\operatorname{sign}(Z) = \lim_{k\to\infty} Z_k;$$

see [11].

The sign function can be used to solve GAREs when applied to the corresponding matrix pencil; see [12, 5]. Pursuing this approach for the GABE, we can apply iteration (5) to the matrix pencil $H - \lambda K$ in (2). Now, exploiting the block-triangular structure of $H - \lambda K$, we obtain from (5) the following *generalized Newton iteration* for the computation of $K\operatorname{sign}\left(K^{-1}H\right)$:

$$\begin{aligned} A_0 \leftarrow A, \quad A_{k+1} &\leftarrow \tfrac{1}{2}\left(\tfrac{1}{c_k}A_k + c_k E A_k^{-1} E\right), \\ G_0 \leftarrow G, \quad G_{k+1} &\leftarrow \tfrac{1}{2}\left(\tfrac{1}{c_k}G_k + c_k E A_k^{-1} G_k A_k^{-T} E^T\right), \end{aligned} \qquad k = 0, 1, 2, \ldots. \tag{6}$$

In order to accelerate the convergence, we can use the following generalization of the determinantal scaling proposed in [13]:

$$c_k := |\det(K^{-1}H_k)|^{1/(2n)} = |\det(A_k)/\det(E)|^{1/n}$$

A suitable stopping criterion for the iteration is to stop when

$$\|A_{k+1} - A_k\|_F \le \tau \cdot \|A\|_F,$$

where τ is a tolerance threshold. In order to avoid stagnation we choose $\tau = \sqrt{\varepsilon}$, with ε as the machine precision, and perform 1–3 additional iterations once this criterion is satisfied. Due to the quadratic convergence of the Newton iteration (5), this is usually enough to achieve the attainable accuracy.

At convergence, after $\bar{k}$ iterations, the solution of (1) can be obtained from the full-rank linear least-squares (LLS) problem (see [13, 11, 5])

$$\begin{bmatrix} G_{\bar{k}} \\ E^T - A_{\bar{k}}^T \end{bmatrix} \hat{X} = \begin{bmatrix} A_{\bar{k}} + E \\ 0_n \end{bmatrix}, \tag{7}$$

and $X = \hat{X}E^{-1}$.

A traditional implementation of the iteration in (6) could proceed by first decomposing A_k into triangular factors L and U via the LU factorization, then solving the linear system EA_k^{-1}, and finally computing three matrix products: $(EA_k^{-1}) \cdot G$, $((EA_k^{-1})G) \cdot (EA_k^{-1})^T$, and $(EA_k^{-1}) \cdot E$. Without taking advantage of the symmetric structure of G and some of the (partial) results, this requires $(26/3)n^3$ flops per iteration. Solving the LLS problem and the linear system in the last stage adds $(22/3)n^3$ flops to this figure. Storage space for 8 square matrices of order n is necessary (indeed, only 6 of these matrices are needed during the iteration and the other two come from the need to solve the LLS).

Overall, the computational cost of the sign function method is in most cases similar to that of an approach based on the QZ algorithm. A rough estimate of the latter method offers a cost of $104n^3$ flops so that 11–12 steps of the sign iterative scheme (a usual number of iterations required for convergence) become as expensive. Also, the storage cost of both approaches is similar. Nevertheless, one major advantage of the sign function approach comes from its easy and efficient parallelization which enables the solution of large-scale equations which cannot be dealt with via the QZ algorithm. Another advantage is the possibility to exploit the low rank structure of G in the factorized form (3)—this will be described in Section 5.

4 An efficient iterative scheme

The computational cost of the sign iteration in (6) can be reduced by exploiting two factors. First, the matrix E is not modified inside the iteration; and second, in case E would present an structure with $\mathcal{O}(n)$ nonzero entries, the cost of the iteration would be much lower.

In order to achieve a simpler structure for E, we propose to transform the GABE into

$$\hat{A}^T\hat{X}\hat{E} + \hat{E}^T\hat{X}\hat{A} - \hat{E}^T\hat{X}\hat{G}\hat{X}\hat{E} = 0, \tag{8}$$

where

$$E = U\hat{E}V^T \tag{9}$$

is a decomposition of E into two orthogonal matrices U and V of order n and a square bidiagonal matrix $\hat{E}$ also of order n. This is the first stage, e.g., in the traditional procedure for computing the singular value decomposition of a matrix. Then,

$$\hat{A} = U^T AV, \quad \hat{G} = U^T GU, \quad \text{and } \hat{X} = U^T XU. \tag{10}$$

The computation of the decomposition in (9) and the corresponding transformations in (10) add to $(32/3)n^3 + 2m^2n$ flops, that is, less than the cost of two sign function iterations implemented in the traditional manner. On the other hand, let us look back at iteration (6), applied to the transformed equation (8) where $E := \hat{E}$ is now bidiagonal. Now, we can implement a step of the iteration by first explicitly inverting A_k, then obtaining the "special" product $E \cdot A_k^{-1}$ (with a computational cost of $\mathcal{O}(n^2)$ flops, which is negligible compared with the cubic cost of most other computations), followed by the computation of the general matrix products $(EA_k^{-1}) \cdot G$, $((EA_k^{-1})G) \cdot (EA_k^{-1})^T$ and the special product $(EA_k^{-1}) \cdot E$. Overall, the computational cost of the iteration becomes $6n^3$ flops which yields a reduction of $(8/3)n^3$ flops per step when compared with the traditional implementation. Furthermore, the amount of storage necessary for this modified scheme is not increased.

5 Solving the factored GABE

Next, consider the factored GABE in (3). We can rewrite the iteration for G_k as follows:

$$B_0 \quad := \quad B, \quad B_{k+1} := \frac{1}{\sqrt{2c_k}} \left[B_k, \; c_k E A_k^{-1} B_k\right],$$

so that $G_k = B_k B_k^T$. Although for $m \ll n$ this iteration is much cheaper than the one for G_k during the initial steps, this advantage is lost as the iteration advances since the number of columns in B_k is doubled at each step. This can be avoided by applying a similar technique to the one used in [14] for the factorized solution of generalized Lyapunov equations.

Let $B_k \in \mathbb{R}^{n \times m_k}$, and consider the rank-revealing QR (RRQR) factorization [15]

$$B_{k+1}^T = \frac{1}{\sqrt{2c_k}} \begin{bmatrix} B_k^T \\ c_k B_k^T A_k^{-T} E^T \end{bmatrix} = Q_k R_k \Pi_k, \quad R_k = \begin{bmatrix} \hat{R}_k \\ 0 \end{bmatrix},$$

where Q_k is orthogonal, Π_k is a permutation matrix, and R_k is upper triangular with $\hat{R}_k \in \mathbb{R}^{r_k \times n}$ of full row-rank. Then, we can use as the new iterate

$$B_{k+1} = \Pi_k^T \hat{R}_k^T$$

and, on convergence, after $\bar{k}$ iterations we will obtain the solution of (3) from $G_{\bar{k}} = B_{\bar{k}} B_{\bar{k}}^T$, the LLS in (7), and $X = \hat{X} E^{-1}$.

Let us analyze the cost of this factored iteration, combined with the modification introduced in the previous section to reduce the cost. In general, $r_k \ll n$ for all the iterations and the cost of the factorized iteration becomes $2n^3 + \mathcal{O}(n^2)$ flops, where the cubic part comes from computing the matrix inverse via, e.g., Gaussian elimination.

This implies a major reduction of the computational cost of the iteration. However, this reduction is balanced by the lower performance of the operations involved in the iteration. The required workspace also depends on the numerical rank of X and is about $6n^2 + 4n\max(r_k)$ numbers.

The variants of the GABE solvers that we have described in the previous sections are basically composed of traditional matrix computations such as matrix (LU and QR) factorizations, solution of triangular linear systems, matrix product, and matrix inversion. All these operations can be performed employing the routines in parallel linear algebra libraries for distributed-memory computers as ScaLAPACK and PLAPACK [16, 17]. The efficacy of the implementations in these libraries carries on to those of our GABE solvers. Here we employ the parallel kernels in ScaLAPACK to implement our solvers.

6 Experimental results

All the experiments presented in this section were performed on a cluster of 30 nodes using IEEE double-precision floating-point arithmetic (machine precision $\varepsilon \approx 2.2204 \times 10^{-16}$). Each node consists of an Intel Xeon processor at 2.4 GHz with 1 GByte of RAM. We employ a BLAS library specially tuned for this processor that achieves around 3800 Mflops/sec. (millions of flops per second) for the matrix product (routine `DGEMM` in GotoBLAS; `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 μs and a bandwidth of 1.4 Gbits/sec.

We include implementations of the following five algorithms in the comparison:

– `ggcbsh`. A GABE solver based on the QZ algorithm.

– `ggcbne_v1`. The iterative GABE solver as in (6).

– `ggcbne_v2`. The iterative GABE solver based on the initial transformation of the equation so that E is reduced to bidiagonal form. The matrix A_k is inverted in this variant using the LU factorization with partial pivoting.

– `ggcbne_v3`. The same algorithm as the previous one but the matrix A_k is now directly inverted using an efficient parallel procedure based on Gauss-Jordan transformations [18]. Both `ggcbne_v2` and `ggcbne_v3` present the same computational cost.

– `ggcbnc_v3`. The iterative factored GABE solver based on the initial transformation of the equation so that E is reduced to bidiagonal form and the iteration for G_k is replaced by that for B_k. Matrix A_k is again directly inverted using Gauss-Jordan transformations.

Example	δ	ggcbsh	ggcbne_v1	ggcbne_v2	ggcbne_v3	ggcbnc_v3
1	1.0e−4	9.7e−15	1.7e−14	2.3e−14	8.8e−15	1.5e−14
2	1.0e−4	3.6e−16	3.5e−12	3.5e−12	4.3e−12	4.3e−12
2	1.0e−2	1.6e−15	1.9e−13	1.1e−13	9.5e−14	9.5e−14

Table 1: Numerical Performance of the different GABE solvers.

6.1 Numerical performance

The experiments in this subsection were performed using MATLAB® 7.0.4.352 (R14). Here we analyze the numerical performance of the GABE solvers as a *stabilizing tool* for linear dynamical systems. In other words, our goal is to find the feedback law $u(t) = -Fx(t)$, with $F = B^T X_* E \in \mathbb{R}^{m\times n}$, which stabilizes the system in (4) by solving the corresponding GABE; see, e.g., [9]. We employ the following two examples in the evaluation:

Example 1. This is Example 4.3 from the ARE benchmark collection in [19] and corresponds to a model of a string consisting of coupled springs, dashpots, and masses with $n = 60$ states, $m = 2$ inputs, and 60 outputs.

Example 2. This model represents an unstable RLC system with $n = 199$ states, $m = 2$ inputs, and 2 outputs.

Since in both examples there is one pole very close to the imaginary axis, posing an ill-conditioned problem for the sign function solvers, we overcome this difficulty by shifting the set of eigenvalues of the matrix pencil $A - \lambda E$ as $\bar{A} - \lambda E$, where $\bar{A} = A + \delta E$. Thus, the actual GABE that is solved is

$$\bar{A}^T XE + E^T X\bar{A} - E^T XBB^T XE = 0.$$

The solution to this equation yields the stabilizing feedback matrix $F = B^T XE$.

Table 1 reports the normalized residual

$$\mathcal{R}_1(X_*) := \frac{\|\bar{A}^T X_* E + E^T X_* \bar{A} - E^T X_* BB^T X_* E\|_1}{\|X_*\|_1},$$

for the solutions X_* computed via the GABE solvers for Example 1 with $\delta = 1.0e - 4$, and Example 2 with $\delta = 1.0e - 4$ and $1.0e - 2$. For the second case the results illustrate that the routine `ggcbsh` yields a smaller residual, while, for the two other cases, all the residuals are similar. In spite of these minor differences among the residuals obtained by the different solvers, stabilizing solutions were obtained by applying all the iterative solvers in all three cases.

In the following subsections we analyze the parallelism and scalability of the different variants of the GABE solver. Unless otherwise stated all the results hereafter correspond to 10 iterations of the schemes. A parallel version of the routine `ggcbsh` is not included in the comparison because presently there is no parallel implementation of the QZ algorithm. Following the convention in ScaLAPACK, we add the prefix `pd-` to the names of our "p"-arallel routines (using "d"-ouble precision arithmetic).

6.2 Parallel performance

Our first experiment reports the execution time of the parallel routines for a GABE of dimension $n = 2400$ using n_p=1, 2, 4, 6, 8, 10, and 12 processors. This is about the largest size we could solve on a single node of the platform considering the number of data matrices involved, the amount of workspace necessary for the computations, and the size of the RAM per node. The execution of the parallel algorithm on a single node is likely to require a higher time than that of a serial implementation of the algorithm (using, e.g., LAPACK and BLAS); however, at least for such problem dimension, our experience tells this overhead is negligible compared with the overall execution time.

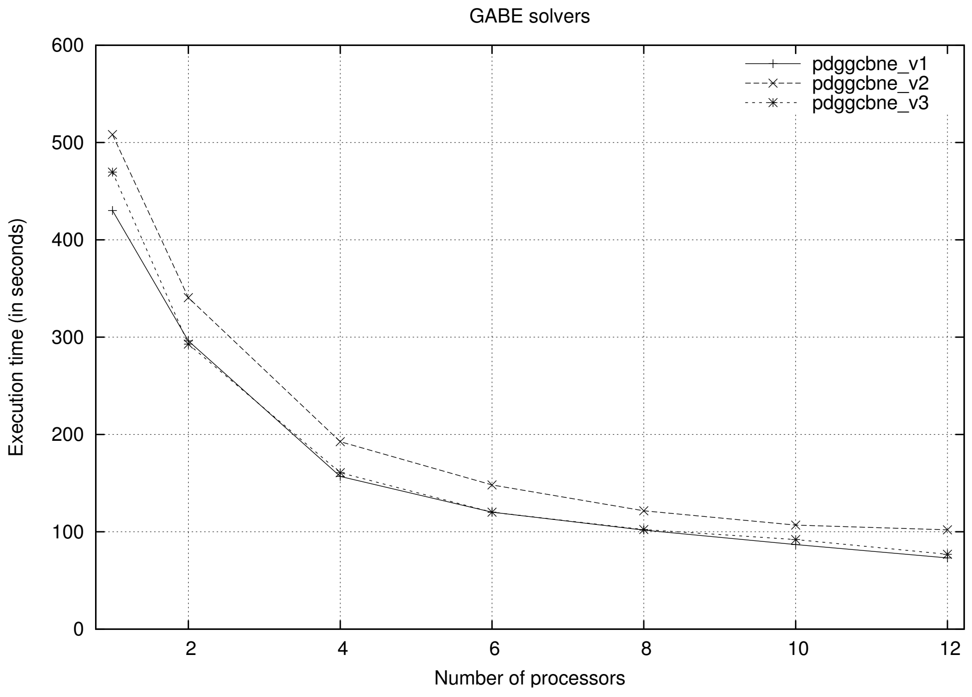

Figure 1: Parallel performance of the `pdggcbne_v1`, `pdggcbne_v2`, and `pdggcbne_v3` variants of the GABE solver on the parallel cluster.

Figure 1 reports a notable reduction in the execution time when a small number of processors is employed, from about 7 minutes to slightly more than 1. Also, for such a small problem, using more than 12 processors does not achieve a significant reduction in the execution time.

Table 2 reports the speed-up of variants `pdggcbne_v1`, `pdggcbne_v2`, and `pdggcbne_v3`. Efficiencies around to 72–80% and 66–73% are attained on 2 and 4 processors, respectively. Naturally, these values decrease as n_p gets larger: while the system dimension is fixed, the problem size per node is reduced, and so is the amount of computations per processor and the opportunity for parallelism.

variant \ n_p	2	4	6	8	10	12
`pdggcbne_v1`	1.45	2.74	3.58	4.23	4.96	5.88
`pdggcbne_v2`	1.49	2.64	3.43	4.18	4.76	4.98
`pdggcbne_v3`	1.60	2.92	3.91	4.59	5.11	6.11

Table 2: Speed-up of the `pdggcbne_v1`, `pdggcbne_v2`, and `pdggcbne_v3` variants of the GABE solver on the parallel cluster.

The results in the previous experiment cannot be used to judge the efficacy of the different variants as a numerical tool to solve the GABE: although variants `pdggcbne_v2`/ `pdggcbne_v3` present a higher fixed computational cost than `pdggcbne_v1`, corresponding to the initial transformation of the GABE into an equation with bidiagonal E, they are also expected to have a lower time cost per iteration. Therefore, a fair comparison should take into account the actual number of iterations required for convergence, which depends on the problem data. We next evaluate the efficacy of these approaches using 1 to 30 processors and different problem sizes, varying from 2400 to 13145. As `pdggcbne_v2` and `pdggcbne_v3` perform the same (number of) computations, but the latter yields better performance, we do not include `pdggcbne_v2` in the comparison.

Table 3 illustrates the minimum number of iterations required for `pdggcbne_v3` to exhibit a lower execution time than `pdggcbne_v1`, and the difference in iteration time between `pdggcbne_v1` and `pdggcbne_v3` (as Δ). Although for one processor and a problem size of 2400, 16 iterations are required to compensate for the higher initial cost in `pdggcbne_v3`, 10–12 iterations are sufficient in the remaining cases. The table also shows that, as the problem size increases, also does the difference between the iteration times of the two variants. In particular, on 32 processors this difference raises from 0.74 seconds for the smallest problem size to to 62.97 seconds for the largest problem dimension. We can therefore conclude that `pdggcbne_v3` will be in general faster than `pdggcbne_v1` when at least 10–12 iterations are required to meet the stopping criterion, and that the difference in favor of the former will enlarge with the ratio problem size/processor.

We next analyze the execution time of the factored variant `pdggcbnc_v3` for a factored GABE of dimension $n = 2400$, with m equal 1%, 0.5%, and 0.1% of that value, using 1, 2, 4, 6, 8, 10, and 12 processors. (We choose m to be much lower than n to mimic a real linear dynamical system, as those arising in stabilization and model reduction problems.) We include variant `pdggcbne_v3` in the comparison as a reference. Figure 2 reports a reduction in the execution time from 7–12 minutes to 1–2 minutes, depending on the specific problem dimensions and variant. Again, for such a small problem, using more than 12 processors does not achieve a significant reduction in the execution time. The results also illustrate that `pdggcbnc_v3` presents a lower execution time than `pdggcbne_v3` when $m = 0.1\% \cdot n$, and therefore is a more efficient tool to solve the GABE. For the other two values of m, `pdggcbne_v3` is still the approach to be preferred. In case solution factors are required, these still have to be calculated from the computed

	1	4	9	16	25	30
2400	16 $\Delta = 7.16$	12 $\Delta = 3.04$	10 $\Delta = 1.97$	11 $\Delta = 1.20$	11 $\Delta = 0.90$	12 $\Delta = 0.74$
4800		12 $\Delta = 20.33$	10 $\Delta = 11.43$	10 $\Delta = 7.20$	10 $\Delta = 5.21$	11 $\Delta = 4.36$
7200			11 $\Delta = 32.28$	11 $\Delta = 21.09$	10 $\Delta = 16.19$	12 $\Delta = 11.61$
9600				10 $\Delta = 45.36$	11 $\Delta = 31.22$	11 $\Delta = 27.40$
12000					11 $\Delta = 58.62$	11 $\Delta = 49.90$
13145						11 $\Delta = 62.97$

Table 3: Comparison between variants `pdggcbne_v1` and `pdggcbne_v3` of the GABE solver for n_p=1, 4, 9, 16, 25 and 30, and problems sizes ranging from 2400 to 13145. Each entry displays the number of iterations for which `pdggcbne_v1` and `pdggcbne_v3` (roughly) present the same execution time; Δ stands for the difference in time per iteration (in seconds) between `pdggcbne_v1` and `pdggcbne_v3` in favor of the latter.

GABE solution if `pdggcbne_v3` is used.

Table 4 reports the speed-up of the factored variant, `pdggcbnc_v3`, for the different values of m. Efficiencies close to 90% are obtained on 2 processors while these values drop to 73–80%, depending on the value of m, on 4 processors. Again, the efficiency further decreases as the number of processors gets larger.

variant \ n_p	2	4	6	8	10	12
`pdggcbnc_v3 1%`	1.81	3.23	4.56	5.51	5.91	6.88
`pdggcbnc_v3 0.5%`	1.78	3.14	4.42	5.24	5.55	6.41
`pdggcbnc_v3 0.1%`	1.74	2.95	4.19	4.82	4.92	5.50

Table 4: Speed-up of the `pdggcbne_v3` and `pdggcbnc_v3` variants of the GABE solver on the parallel cluster.

6.3 Scalability

We finally evaluate the scalability of the parallel algorithms. In order to keep the problem size per node constant, we fix the problem dimensions to $n/\sqrt{n_p} \approx 2400$, with m=1%, 0.5%, and 0.1% of n. Figure 3 reports the Mflops/sec. per node of the parallel routines, showing a high degree of scalability, as there is only a small decrease in the performance of the algorithms when n_p is increased while the problem dimension per node remains

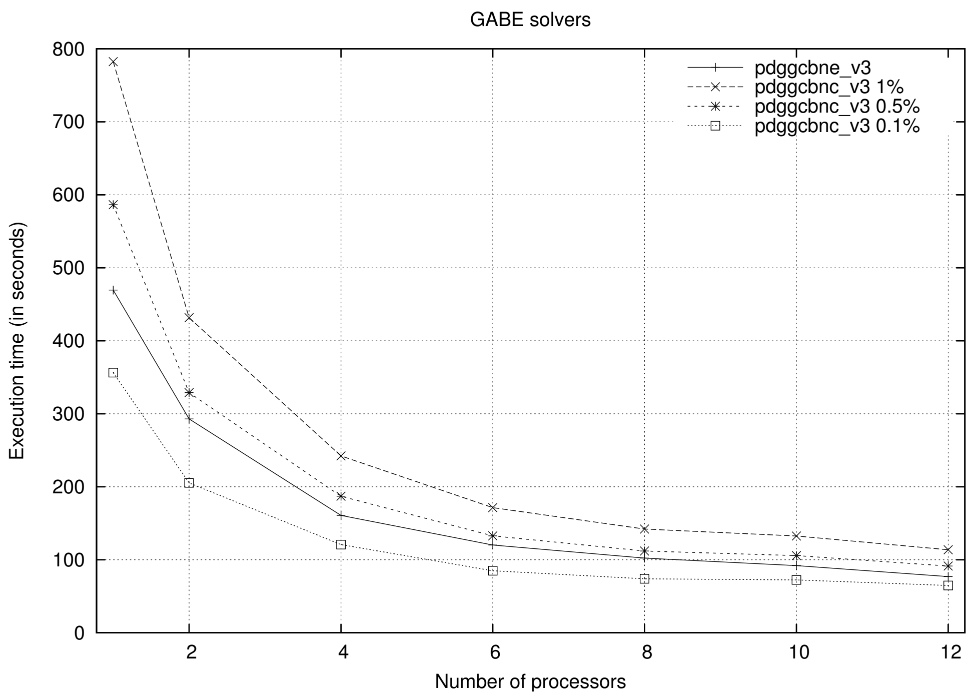

Figure 2: Parallel performance of the `pdggcbne_v3` and `pdggcbnc_v3` variants of the GABE solver on the parallel cluster.

fixed. Specifically, in going from a serial execution to a parallel one, a moderate loss of performance is experienced, mainly due to the communication overhead. The Mflops/sec. rate is then maintained almost constant except for the $n_p = 30$ case, where a minor decrease is encountered. This can be due to the use of a non-square logical processor topology in this case. As expected, the `pdggcbnc_v3` variant delivers a lower Mflops/sec. rate than the other ones. This is due to the nature of the operations used by this variant (rank-revealing QR factorizations) being less efficient than those used by `pdggcbne_v1`, `pdggcbne_v2`, and `pdggcbne_v3`. However, as the `pdggcbnc_v3` variant usually performs fewer operations, it is still possible to obtain lower execution times with this variant.

7 Conclusions

We have presented a new matrix sign function-based iterative scheme to obtain the solution of the GABE. Two major modifications of the basic algorithm are proposed: An initial transformation of the equation reduces the computational cost of the iteration by 30%, at the expense of an increase of the initial number of operations. This variant has been shown to outperform the basic algorithm when a moderate number of iterations is required for convergence; the actual number of iterations is inversely proportional to

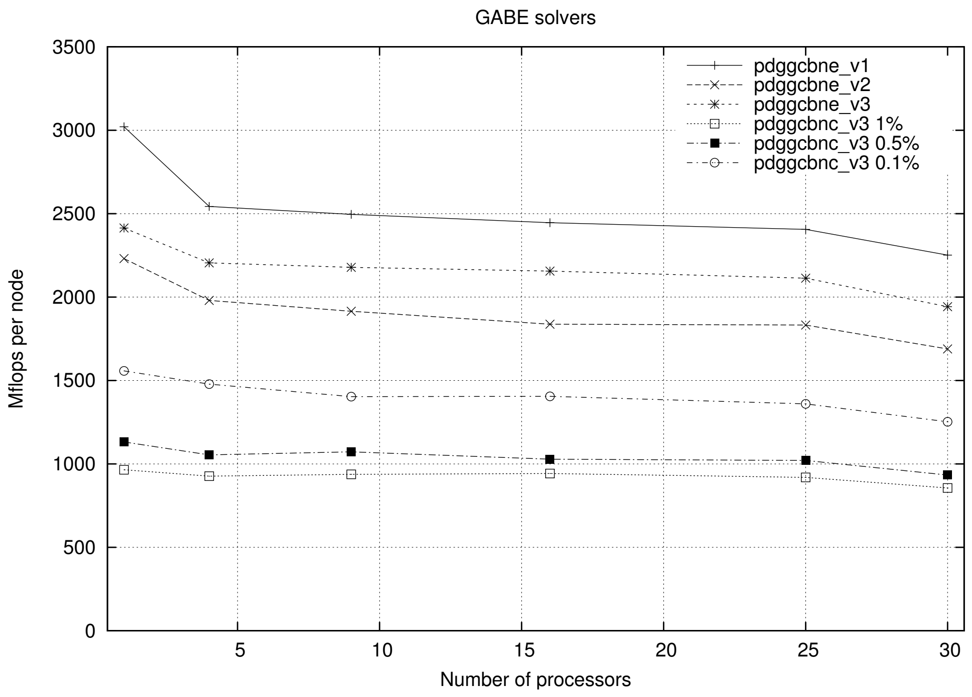

Figure 3: Scalability of all the variants of the GABE solver on the parallel cluster.

the ratio problem size/processor. A different modification provides a factorized version of the iteration in case the matrix in the quadratic term of the equation is given in factored form. Our numerical experiments show the benefits of using this second approach when the dimension m of the problem is much lower than the order of the equation (a typical case in real applications!).

The variants resulting from our approach can be easily parallelized using the kernels in ScaLAPACK (or in any other parallel linear algebra library), yielding efficient and scalable algorithms. The use of these algorithms thus enables the solution of GABEs with dimension n in the order of thousands on parallel distributed-memory computers.

References

[1] V. Ionescu, C. Oară, M. Weiss, Generalized Riccati theory and robust control. A Popov function approach, John Wiley & Sons, Chichester, 1999.

[2] K. Zhou, J. Doyle, K. Glover, Robust and Optimal Control, Prentice-Hall, Upper Saddle River, NJ, 1996.

[3] K. Zhou, G. Salomon, E. Wu, Balanced realization and model reduction for unstable systems, Int. J. Robust Nonlinear Control 9 (3) (1999) 183–198.

[4] M. Kamon, F. Wang, J. White, Recent improvements for fast inductance extraction and simulation [packaging], in: Proc. of the IEEE 7th Topical Meeting on Electrical Performance of Electronic Packaging, 1998, pp. 281–284.

[5] V. Sima, Algorithms for Linear-Quadratic Optimization, Vol. 200 of Pure and Applied Mathematics, Marcel Dekker, Inc., New York, NY, 1996.

[6] C. B. Moler, G. W. Stewart, An algorithm for generalized matrix eigenvalue problems, SIAM J. Numer. Anal. 10 (1973) 241–256.

[7] P. Benner, R. Byers, V. Mehrmann, H. Xu, Numerical computation of deflating subspaces of skew-Hamiltonian/Hamiltonian pencils, SIAM J. Matrix Anal. Appl. 24 (1) (2002) 165–190.

[8] G. Henry, R. van de Geijn, Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: myths and reality, SIAM J. Sci. Comput. 17 (1997) 870–883.

[9] S. Barrachina, P. Benner, E. Quintana-Ortí, Parallel solution of large-scale algebraic Bernoulli equations via the matrix sign function method, in: Proc. of the 2005 Int. Conference on Parallel Processing (ICPP-05), 2005, pp. 189–193.

[10] P. Lancaster, L. Rodman, The Algebraic Riccati Equation, Oxford University Press, Oxford, 1995.

[11] J. Roberts, Linear model reduction and solution of the algebraic Riccati equation by use of the sign function, Internat. J. Control 32 (1980) 677–687, (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

[12] J. Gardiner, A. Laub, A generalization of the matrix-sign-function solution for algebraic Riccati equations, Internat. J. Control 44 (1986) 823–832.

[13] R. Byers, Solving the algebraic Riccati equation with the matrix sign function, Linear Algebra Appl. 85 (1987) 267–279.

[14] P. Benner, E. Quintana-Ortí, Solving stable generalized Lyapunov equations with the matrix sign function, Numer. Algorithms 20 (1) (1999) 75–100.

[15] T. Chan, Rank revealing QR factorizations, Linear Algebra Appl. 88/89 (1987) 67–82.

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

[17] R. van de Geijn, Using PLAPACK: Parallel Linear Algebra Package, MIT Press, Cambridge, MA, 1997.

[18] E. Quintana-Ortí, G. Quintana-Ortí, X. Sun, R. van de Geijn, A note on parallel matrix inversion, SIAM J. Sci. Comput. 22 (2001) 1762–1771.

[19] J. Abels, 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.net`.