COMPUTATIONAL EXPERIENCE IN SOLVING LARGE LINEAR MATRIX EQUATIONS FOR AUTOMATIC CONTROL

Vasile Sima [*,1]

[*] *National Institute for Research & Development in Informatics*
Bd. Mareşal Averescu, Nr. 8–10, 011455 Bucharest 1, Romania

Abstract: State-of-the-art, uni-processor linear matrix equation solvers for automatic control computations are investigated and compared for various problem sizes. General-purpose SLICOT solvers are the most efficient ones for small-size problems, but they cannot compete for larger problems with specialized solvers designed for certain problem classes. *Copyright ©2004 IFAC*

Keywords: Computer-aided control system design, Large-scale systems, Linear control systems, Lyapunov equation, Numerical algorithms, Numerical solutions

1. INTRODUCTION

Numerical algorithms are increasingly used to model, simulate, and optimize industrial, economical, and biological processes. Human-made systems are heavily dependent on computer technology and automatic control concepts and algorithms. Control systems analysis and design procedures often require the solution of general or special linear or quadratic matrix equations. Examples are: invariant or deflating subspaces of matrices or matrix pairs, block-diagonalization and computation of matrix functions, controllability and observability Gramians, Hankel singular values, model and controller reduction, Newton-type algorithms for linear-quadratic optimization, condition estimation for eigenvalue problems and linear or quadratic matrix equations, etc. There is a huge amount of theoretical results available both in systems and control, as well as in the linear algebra literature devoted to matrix equations and related topics. There are also many associated software implementations, both commercial (e.g., in MATLAB [2] (MathWorks, 1998, 1999)), copyrighted freeware (e.g., in the SLICOT Library (Benner *et al.*, 1999; Van Huffel and Sima, 2002; Van Huffel *et al.*, 2004)), or in the public domain (e.g., in Scilab (Gomez, 1999)). The reliability, efficiency, and functionality of various solvers differ significantly from package to package.

Although numerical algorithms for linear matrix equations in control theory were published since 1960, this is still a very hot research topic. The challenge for solving larger and larger equations has not yet been fully answered. The proposed techniques are usually not general enough. The comparative studies of the numerical techniques are also limited. It is the purpose of this paper to investigate the performances of several powerful solvers for linear matrix equations.

The capabilities and limitations of the general-purpose solvers available in the SLICOT Library and MATLAB are studied, in comparison with some specialized solvers. SLICOT Library (**S**ubroutine **L**ibrary **I**n **CO**ntrol **T**heory) provides Fortran 77 implementations of many numerical algorithms in systems and control theory, as well as standardized interfaces (gateways) to MATLAB and Scilab. Built around a nucleus of basic numerical linear algebra subroutines from the state-of-the-art software packages LAPACK (Anderson *et al.*, 1999), BLAS (Dongarra

[1] This work was supported in part by the Project PN 03 13 01 03: *Efficient Algorithms for Analysis and Design of Large Scale Control Systems*, I.C.I., Bucharest

[2] MATLAB is a registered trademark of The MathWorks, Inc.

et al., 1988, 1990; Lawson *et al.*, 1979), and their counterparts for distributed memory computers, e.g., ScaLAPACK and PBLAS, this library enables to exploit the potential of modern high-performance computer architectures. The SLICOT solvers for linear matrix equations offer improved efficiency, reliability, and functionality over the corresponding solvers in other computer-aided control system design packages.

2. SLICOT LINEAR MATRIX EQUATION SOLVERS CAPABILITIES

The extended functionality of SLICOT solvers is partly illustrated by the following list of equations solvable by SLICOT codes (Sima and Benner, 2003; Slowik *et al.*, 2004).

- Continuous- and discrete-time Sylvester equations:

$$\text{op}(A)\,X \pm X\,\text{op}(B) = \sigma C; \tag{1}$$
$$\text{op}(A)\,X\,\text{op}(B) \pm X = \sigma C; \tag{2}$$

- Continuous- and discrete-time Lyapunov equations:

$$\text{op}(A)^T X + X\,\text{op}(A) = \sigma C; \tag{3}$$
$$\text{op}(A)^T X\,\text{op}(A) - X = \sigma C; \tag{4}$$

- Stable non-negative definite continuous- and discrete-time Lyapunov equations:

$$\text{op}(A)^T X + X\,\text{op}(A) = -\sigma^2\,\text{op}(D)^T\,\text{op}(D)\,; \tag{5}$$
$$\text{op}(A)^T X\,\text{op}(A) - X = -\sigma^2\,\text{op}(D)^T\,\text{op}(D)\,; \tag{6}$$

- Generalized Sylvester equation:

$$\begin{aligned} AX - YB &= \sigma G, \\ EX - YF &= \sigma H; \end{aligned} \tag{7}$$

  or the "transposed" equation

$$\begin{aligned} A^T X + E^T Y &= \sigma G, \\ XB^T + YF^T &= -\sigma H; \end{aligned} \tag{8}$$

- Generalized continuous- and discrete-time Lyapunov equations:

$$\text{op}(A)^T X\,\text{op}(E) + \text{op}(E)^T X\,\text{op}(A) = \sigma C; \tag{9}$$
$$\text{op}(A)^T X\,\text{op}(A) - \text{op}(E)^T X\,\text{op}(E) = \sigma C; \tag{10}$$

- Generalized stable continuous- and discrete-time Lyapunov equations:

$$\text{op}(A)^T X\,\text{op}(E) + \text{op}(E)^T X\,\text{op}(A) = -\sigma^2\,\text{op}(D)^T\,\text{op}(D)\,; \tag{11}$$
$$\text{op}(A)^T X\,\text{op}(A) - \text{op}(E)^T X\,\text{op}(E) = -\sigma^2\,\text{op}(D)^T\,\text{op}(D)\,; \tag{12}$$

where the notation $\text{op}(M)$ denotes either the matrix M, or its transpose, M^T, A, B, $\text{op}(D)$, E, and F, are $n \times n$, $m \times m$, $m \times n$, $n \times n$, and $m \times m$ given matrices, respectively, C, G, and H are given matrices of appropriate dimensions, X and Y are unknown matrices of appropriate dimensions, and σ is a scaling factor, usually equal to one, but possibly set less than one, in order to prevent overflow in the solution matrix.

Let $\mathcal{E}(\mathcal{D},\mathcal{U}) = \mathcal{R}$ be a shorthand notation for any of the above equations, where $\mathcal{E}$, $\mathcal{D}$, $\mathcal{U}$, and $\mathcal{R}$ denote the corresponding equation formula, data, unknowns, and right hand side term, respectively. For general matrices, the solution is obtained by a *transformation method* (see, e.g., (Sima, 1996, page 144)). Specifically, the data $\mathcal{D}$ are transformed to some simpler forms, $\widetilde{\mathcal{D}}$ (usually corresponding to the real Schur form (RSF) of A, or generalized RSF of a matrix pair), the right hand side term is transformed accordingly to $\widetilde{\mathcal{R}}$, the *reduced equation*, $\mathcal{E}(\widetilde{\mathcal{D}},\widetilde{\mathcal{U}}) = \widetilde{\mathcal{R}}$, is solved in $\widetilde{\mathcal{U}}$, and finally, the solution of the original equation is recovered from $\widetilde{\mathcal{U}}$.

The methods implemented in SLICOT are basically the following: the Schur method (also known as Bartels–Stewart method) (Bartels and Stewart, 1972) for Sylvester equations (for A, B general, or in RSF), or Lyapunov equations (for A general, or in RSF), with the variant from (Barraud, 1977) for the discrete-time case; the Hessenberg-Schur method in (Golub *et al.*, 1979) for standard Sylvester equations, i.e., with $\text{op}(M) = M$ (for A, B general, or at least one of A or B in RSF, and the other one in Hessenberg or Schur form, both either upper or lower); Hammarling's variant (Hammarling, 1982) of the Bartels–Stewart method for stable Lyapunov equations; and extensions of the above methods for generalized Sylvester (Kågström and Poromaa, 1996) and Lyapunov equations (Penzl, 1998).

The ability to work with the $\text{op}(\cdot)$ operator is important in many control analysis and design problems. For instance, the controllability Gramians can be defined as solutions of stable Lyapunov equations with $\text{op}(A) = A^T$, while observability Gramians can be defined as solutions of stable Lyapunov equations with $\text{op}(A) = A$. When both controllability and observability Gramians are needed (e.g., in model reduction computations), then the same real Schur form of A can be used by a solver able to cope with $\text{op}(\cdot)$, and this significantly improves the efficiency.

The solvers for stable Lyapunov equations directly compute the Cholesky factor U of the solution matrix X, i.e., $X = \text{op}(U)^T\,\text{op}(U)$. Whenever feasible, the use of the stable solvers instead of the general ones is to be preferred, for several reasons, including the following: • the matrix product $\text{op}(D)^T\,\text{op}(D)$ need not be computed; • definiteness of X is guaranteed. Moreover, often the Cholesky factors themselves are actually needed, e.g., for model reduction or for computing the Hankel singular values of the system.

When solving any matrix equation, it is useful to have estimates of the *problem conditioning* and of the solution accuracy, e.g., *error bounds*. Such measures are returned by several routines of the SLICOT Library (Sima *et al.*, 2000), allowing to assess the quality of the computed solution. This illustrates the increased reliability and functionality of the software available in SLICOT. In contrast, many other control packages could merely compute the solution residual, which can be misleading if the problem is ill-conditioned.

3. SPECIALIZED LINEAR MATRIX EQUATION SOLVERS

The results in (Sima and Benner, 2003; Slowik *et al.*, 2004) and other papers, as well as those included below, show that the high-level MATLAB interfaces to the SLICOT codes offer improved efficiency (at comparable accuracy) over the existing standard software tools. However, the SLICOT solvers do have some limitations, mainly coming from their generality. These solvers cannot compete in terms of efficiency with specialized solvers designed for specific classes of large-scale problems.

Two types of specialized solvers are considered in this investigation: iterative algorithms for stable Lyapunov equations with low rank solutions, and recursive algorithms for quasi-triangular linear matrix equations.

3.1 Iterative algorithms for stable, low rank Lyapunov equations

The approach for solving large-scale Lyapunov equations implemented in the MATLAB package LYAPACK (LYApunov PACKage) (Penzl, 2000) can be applied to structured (or sparse) stable continuous-time equations of the form

$$FX + XF^T = -GG^T, \tag{13}$$

where $F \in \mathbb{R}^{n\times n}$ and $G \in \mathbb{R}^{n\times m}$. In many applications, for instance, model reduction or algebraic Riccati equations, it is sufficient to obtain a factorization of the solution matrix X, $X = ZZ^T$. For solving Riccati equations, it is assumed that the matrix F has the form $F = A - BK^T$, where A and B are the matrices of the system dynamics equation, and K^T is the gain matrix of the optimal regulator.

Besides the limitations imposed by the form of the equation (13) and stability hypothesis, it is also assumed that the number of columns m is small in comparison with n, $m \ll n$, and that the matrix F is structured so that efficient solution of linear systems with coefficient matrices $F - pI_n$, where $p \in \mathbb{C}$, as well as efficient computation of matrix-vector products are possible. Moreover, the order n should be large enough, for instance, $n > 500$, and the equations be sufficiently well-conditioned.

The LYAPACK approach, implemented in the function `lp_lradi`, uses the *low rank Cholesky factor* technique, in combination with *alternate directions* method, abbreviated as LRCF-ADI (Low Rank Cholesky Factor Alternate Directions Iterations). The efficiency of LRCF-ADI depends on certain ADI *shift parameters*, computed by an heuristic algorithm. The low rank Cholesky factor technique is based on the observation that in many problems (13) with $m \ll n$, the eigenvalues of the solution matrix X decay very fast, which suggests the possible existence of very accurate approximations of rank much smaller than n. The ADI iteration for Lyapunov equation (13) is given by

$$\begin{aligned}(F + p_iI_n)X_{i-1/2} &= -GG^T - X_{i-1}(F^T - p_iI_n),\\ (F + \bar{p}_iI_n)X_i^T &= -GG^T - X_{i-1/2}^T(F^T - \bar{p}_iI_n),\end{aligned}$$

for $i = 1, 2, \ldots$, where $X_0 = 0$. Each iteration involves matrix-vector products and solutions of sparse linear systems. The convergence is accelerated using the parameters p_i. This method generates a sequence of matrices X_i which converges often very fast to the solution, provided that the ADI shift parameters, p_i, are chosen in a (sub)optimal way. The efficient implementation of the ADI method replaces the iterates X_i by their Cholesky factors, $X_i = Z_iZ_i^T$. Let $\mathcal{P}_j$ be a real negative number, or a pair of complex conjugated numbers with negative real part. If the matrix $X_i = Z_iZ_i^T$ is generated by a *proper* set of parameters $\{p_1, p_2, \ldots, p_i\} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_i\}$, then X_i is a real matrix. The problem of finding (sub)optimal ADI parameters, $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\ell\}$, is strongly connected to the rational minimax problem applied to the function

$$s_{\mathcal{P}}(t) = \frac{|(t - p_1)\cdot\ \cdots\ \cdot(t - p_\ell)|}{|(t + p_1)\cdot\ \cdots\ \cdot(t + p_\ell)|}.$$

This problem is stated as $\min_{\mathcal{P}} \max_{t\in\sigma(F)} s_{\mathcal{P}}(t)$, where $\sigma(F)$ denotes the spectrum of the matrix F. The implementation of the heuristic technique first generates a discrete set, which approximates the spectrum, using a pair of Arnoldi processes. The first process, acting on the matrix F, produces k_+ *Ritz values* which tend to approximate the eigenvalues farest from the origin. The second process, acting on the matrix F^{-1}, produces k_- Ritz values, approximations of the eigenvalues close to the origin. The set of shift parameters is then chosen as a subset of the Ritz values, as an heuristic, suboptimal solution of the resulting discrete optimization problem.

The use of the LYAPACK package implies that the user writes the specific routines performing operations with matrices F or A, of the form

$$\begin{aligned}&\mathcal{Y} \longleftarrow AY \ \text{ or } \ \mathcal{Y} \longleftarrow A^TY,\\ &\mathcal{Y} \longleftarrow A^{-1}Y \ \text{ or } \ \mathcal{Y} \longleftarrow A^{-T}Y,\\ &\mathcal{Y} \longleftarrow (A + p_iI_n)^{-1}Y \ \text{ or } \ \mathcal{Y} \longleftarrow (A^T + p_iI_n)^{-1}Y,\end{aligned}$$

where $Y \in \mathbb{C}^{n\times t}$, $t \ll n$.

3.2 Recursive algorithms for quasi-triangular linear matrix equations

An approach (Jonsson and Kågström, 2002a, b) which can be applied to all classes of linear matrix equations with quasi-triangular matrices is based on the use of recursive algorithms. The basic idea is to recursively decompose the quasi-triangular matrices in blocks until the obtained equations are small enough for being solved in the very fast cache memory, and to use some "superscalar" computational kernels for equations of small dimensions. The sizes of the blocks are variable, and this fact enables their automatic adaptation to the computational platform used, and the efficient exploitation of the existing memory hierarchies on modern computing machines.

To illustrate, consider the case of a continuous-time Sylvester equation,

$$AX - XB = C, \tag{14}$$

where A and B are $m\times m$ and $n\times n$ matrices, respectively, either upper triangular or in real Schur form. Depending on the values of m and n, three alternative *recursive decompositions* can be considered. One such alternative is illustrated below.

If $1 \le n \le m/2$, A is decomposed by rows and columns, and C is decomposed by rows:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} - \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} B = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}, \tag{15}$$

or, equivalently,

$$\begin{aligned} A_{11}X_1 - X_1B &= C_1 - A_{12}X_2, \\ A_{22}X_2 - X_2B &= C_2. \end{aligned} \tag{16}$$

Two triangular Sylvester equations have been obtained. The second equation is solved in X_2, and after a `GEMM`-type update, $C_1 \leftarrow C_1 - A_{12}X_2$, the first Sylvester equation is solved.

There are three levels of solvers for linear matrix equations. The recursive block solvers are destined to the user. Each of these solvers calls a *sub-system* block solver, when the dimensions m and n of the current subproblem in the recursive decomposition are smaller than a certain block size, `blks`. Finally, each sub-system solver calls a *superscalar kernel* for solving equations with $m, n \le 4$. Besides the advantageous use of the memory hierarchies, the recursive approach allows to consider various forms of parallelism. The major disadvantage of the recursive solvers is that they merely solve "reduced" equations. The initial reduction to the (generalized) real Schur form is not covered. The codes are implemented in the RECSY library, and wrappers to the SLICOT solvers are provided, so that general equations can be solved, and condition estimates can be computed.

4. NUMERICAL RESULTS

Some typical results are graphically illustrated in the figures below. The calculations have been done on a PC computer with a 500 MHz Intel processor, 128 Mb memory and the relative machine precision $\epsilon = 2.22\times 10^{-16}$, using Compaq Visual Fortran V6.5, optimized BLAS provided by MATLAB, and MATLAB 6.5.1 (R13).

One application has a band matrix $A \in \mathbb{R}^{n\times n}$ with 5 nonzero diagonals, obtained by discretization of a partial differential equation, using finite differences on an equidistant grid. The rows and columns of A are further permuted so that a matrix with an even smaller bandwidth is obtained. (The resulting matrix for order 1000 has only 4992 nonzero elements.) The right-hand side matrix has the form BB^T, where $B \in \mathbb{R}^n$. The data matrices have been generated by the LYAPACK example codes `fdm_2d_matrix` and `fdm_2d_vector`. The LYAPACK solver `lp_lradi` is faster than the SLICOT solver `sllyap` for Lyapunov equations of order higher than, say, 270 (and much faster for larger orders); `sllyap`, in turn, is 2–3 (or much more) times faster than MATLAB `lyap`.[3] But for problems of order smaller than 270, `lp_lradi` is slower (possibly much slower) than `sllyap`. Note also that, besides the stability and sparsity requirements, `lp_lradi` also assumes some additional conditions, such as: the solution matrix has a small rank; the equation is quite well-conditioned; the equation order is large enough.

Figures 1, 2 and 3 show the execution times for the solvers `sllyap`, `lyap` and `lp_lradi`, in SLICOT, MATLAB and LYAPACK, respectively, in the ranges $n \le 225$, $196 \le n \le 400$, and $400 \le n \le 1024$, as well as their ratios, taking `sllyap` as a reference. The equations with orders in these ranges could be considered as "small", "medium", and "large", respectively, for the computer used for their solution. The results show that SLICOT routines always outperform MATLAB calculations (for any order), and also specialized solvers, for problems of small size. It should be mentioned that the accuracy is comparable for all these solvers and all equations solved. The results also show the superiority of the `lp_lradi` solver over `sllyap` and, especially, `lyap`, when solving Lyapunov equations of order larger than 300. It should be, however, mentioned that, in contrast with `(sl)lyap`, `lp_lradi` is not a general solver. Its high efficiency is due to the use of the sparse structure of the matrix A in operations like Ab or $A^{-1}b$, where b is a vector.

Figures 4 and 5 present the execution times and their ratios to those for `sllyap` when calling the recursive algorithms for another application. In this case, the

[3] The latest `lyap` version included in MATLAB 7, released in June 2004, is not considered, since it is based on the corresponding SLICOT routines; this version could also be about 20 % slower than `sllyap`.

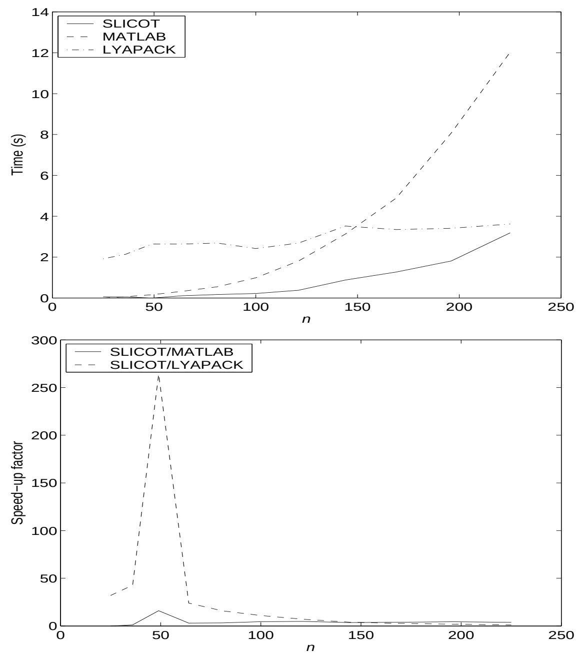

Fig. 1. Application 1, $n \leq 225$. Top: The execution times. Bottom: The speed-up factors.

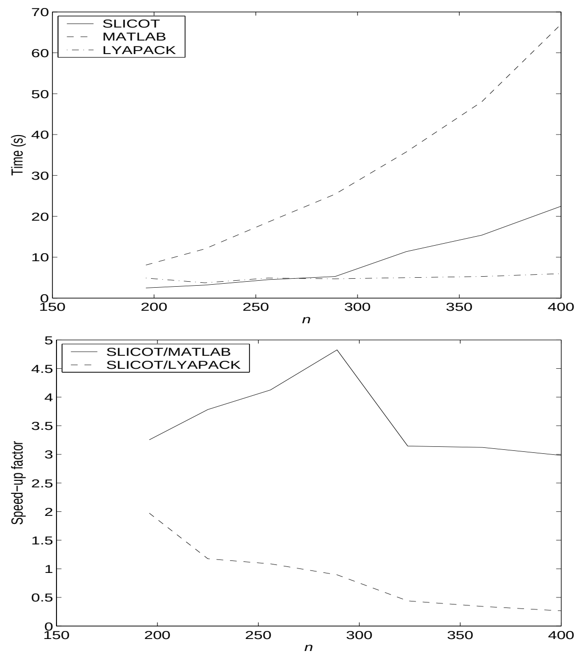

Fig. 2. Application 1, $196 \leq n \leq 400$. Top: The execution times. Bottom: The speed-up factors.

matrix A has been obtained starting from a Jordan form, and applying an orthogonal similarity transformation, which filled-up the matrix with nonzero elements and altered its condition number. Function `lp_lradi` becomes more efficient than `sllyap` for $n \geq 150$.

5. CONCLUSIONS

Various state-of-the-art, uni-processor linear matrix equation solvers for automatic control computations have been investigated and compared for various problem sizes. The results confirm the natural expectation that general-purpose solvers, such as those currently implemented in the SLICOT Library (and, consequently, in MATLAB 7) cannot compete in efficiency, for large-scale problems, with specialized solvers designed for certain problem classes. However, the SLICOT solvers are the most efficient ones for small-size problems. Moreover, they are general solvers and offer extended functionality and broad computational abilities.

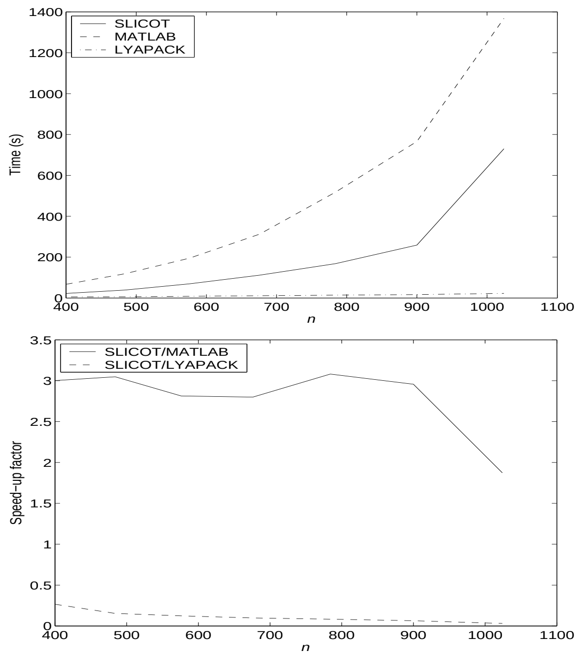

Fig. 3. Application 1, $400 \leq n \leq 1024$. Top: The execution times. Bottom: The speed-up factors.

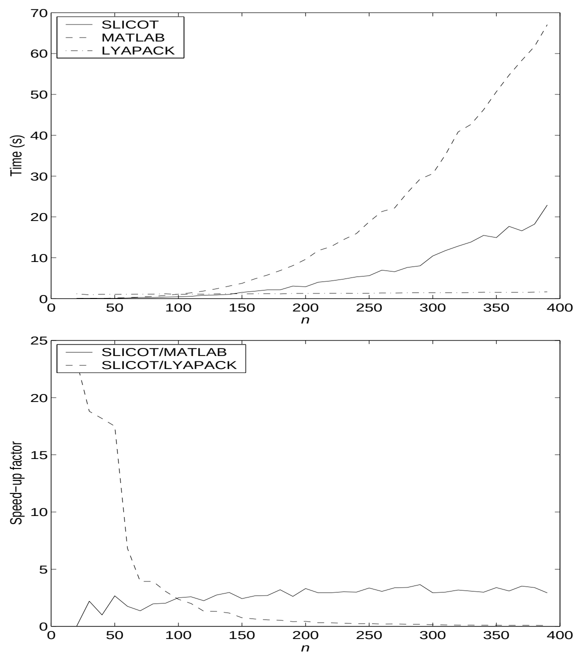

Fig. 4. Application 2, $n \leq 400$, recursive algorithm. Top: The execution times. Bottom: The speed-up factors.

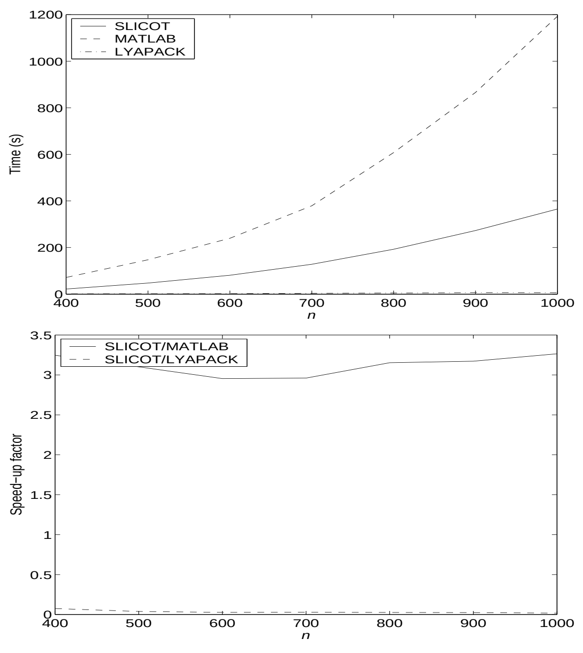

Fig. 5. Application 2, $400 \leq n \leq 1000$, recursive algorithm. Top: The execution times. Bottom: The speed-up factors.

REFERENCES

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen (1999). *LAPACK Users' Guide: Third Edition*. SIAM, Philadelphia.

Barraud, A.Y. (1977). A numerical algorithm to solve $A^T XA - X = Q$. *IEEE Trans. Automat. Control* **AC-22**(5), 883–885.

Bartels, R.H. and G.W. Stewart (1972). Algorithm 432: Solution of the matrix equation $AX + XB = C$. *Comm. ACM* **15**(9), 820–826.

Benner, P., V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga (1999). SLICOT — A subroutine library in systems and control theory. In: *Applied and Computational Control, Signals, and Circuits* (B.N. Datta (Ed.)), Vol. **1**, ch. 10, 499–539. Birkhäuser, Boston.

Dongarra, J.J., J. Du Croz, I.S. Duff and S. Hammarling (1990). Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**(1), 1–17, 18–28.

Dongarra, J.J., J. Du Croz, S. Hammarling and R.J. Hanson (1988). Algorithm 656: An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**(1), 1–17, 18–32.

Golub, G.H., S. Nash and C.F. Van Loan (1979). A Hessenberg-Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control* **AC–24**(6), 909–913.

Gomez, C. (Ed.) (1999). *Engineering and Scientific Computing with Scilab*. Birkhäuser, Boston.

Hammarling, S.J. (1982). Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.* **2**(3), 303–323.

Jonsson, I. and B. Kågström (2002*a*). Recursive blocked algorithms for solving triangular systems—Part I: One-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Softw.* **28**(4), 392–415.

Jonsson, I. and B. Kågström (2002*b*). Recursive blocked algorithms for solving triangular systems—Part II: Two-sided and generalized Sylvester and Lyapunov matrix equations. *ACM Trans. Math. Softw.* **28**(4), 416–435.

Kågström, B. and P. Poromaa (1996). LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs. *ACM Trans. Math. Softw.* **22**(1), 78–103.

Lawson, C.L., R.J. Hanson, D.R. Kincaid and F.T. Krogh (1979). Basic Linear Algebra Subprograms for Fortran usage. *ACM Trans. Math. Softw.* **5**(3), 308–323.

MathWorks (1998). *Control System Toolbox User's Guide*. The MathWorks, Inc., Natick, MA.

MathWorks (1999). *Using MATLAB. Version 5*. The MathWorks, Inc., Natick, MA.

Penzl, T. (1998). Numerical solution of generalized Lyapunov equations. *Advances in Comp. Math.* **8**, 33–48.

Penzl, T. (2000). LYAPACK Users Guide. Techn. Rep. SFB393/00–33. Technische Universität Chemnitz, Sonderforschungsbereich 393. Chemnitz.

Sima, V. (1996). *Algorithms for Linear-quadratic Optimization*. Marcel Dekker, Inc. New York.

Sima, V. and P. Benner (2003). Solving linear matrix equations with SLICOT. In: *Proceedings of the European Control Conference ECC'03*, 1–4 September, 2003. Cambridge, UK. Special Session *Matrix Equations in Systems and Control,* organized by P. Benner and V. Sima, 6 pages.

Sima, V., P. Petkov and S. Van Huffel (2000). Efficient and reliable algorithms for condition estimation of Lyapunov and Riccati equations. In: *Proceedings CD of the Fourteenth International Symposium of Mathematical Theory of Networks and Systems MTNS-2000*, Perpignan, France, June 19–23, 2000. Session CS 2C, 10 pages.

Slowik, M., P. Benner and V. Sima (2004). Evaluation of the linear matrix equation solvers in SLICOT. Tech. Rep. Institut für Mathematik, MA 4-5, Technical University Berlin. Straße des 17. Juni 136, D-10623 Berlin, Germany. To be published.

Van Huffel, S. and V. Sima (2002). SLICOT and control systems numerical software packages. In: *Proceedings of the 2002 IEEE International Conference on Control Applications and IEEE International Symposium on Computer Aided Control System Design, CCA/CACSD 2002*, September 18–20, 2002, Glasgow, U.K., 39–44. Omnipress.

Van Huffel, S., V. Sima, A. Varga, S. Hammarling and F. Delebecque (2004). High-performance numerical software for control. *IEEE Control Systems Magazine* **24**(1), 60–76.