Evaluation of the Linear Matrix Equation Solvers in SLICOT [1]

Martin Slowik[2] Peter Benner[3] Vasile Sima[4]

August 2004

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and supported in part by the DFG Research Center “Mathematics for Key Technologies” (FZT 86), Berlin, and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001-Leuven-Heverlee, BELGIUM. This report is available by anonymous ftp from `wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2004-1.ps.Z`
[2]Institut für Mathematik, MA 4-5, TU Berlin, Straße des 17. Juni 136, D-10623 Berlin, Germany; Email `slowik@math.tu-berlin.de`
[3]Fakultät für Mathematik, TU Chemnitz, D-09107 Chemnitz, Germany; Email: `benner@mathematik.tu-chemnitz.de`
[4]National Institute for Research & Development in Informatics, 011455 Bucharest 1, Romania; Email `vsima@iciadmin.ici.ro`

Abstract

We discuss solvers for Sylvester, Lyapunov, and Stein equations that are available in the SLICOT Library (**S**ubroutine **L**ibrary **I**n **CO**ntrol **T**heory). These solvers offer improved efficiency, reliability, and functionality compared to corresponding solvers in other computer-aided control system design packages. The performance of the SLICOT solvers is compared with the corresponding MATLAB solvers. This note can also serve as a guide to the SLICOT and SLICOT-based MATLAB solvers for Linear Matrix Equations.

Keywords: computer-aided control system design, numerical algorithms, numerical linear algebra, Lyapunov equations, Sylvester equations.

1 Introduction

Systems and control algorithms are widely used to model, simulate, and/or optimize industrial, economical, and biological processes. Systems analysis and design procedures often require the solution of general or special linear or quadratic matrix equations. Many high-level algorithms are based on these low-level kernels. This is particularly true for model reduction methods based on balancing system Gramians (see [2, 22] and the references therein). As these Gramians are given as solutions of Lyapunov or related matrix equations, the applicability of these methods to large-scale problems mainly depends on the efficient solvability of linear matrix equations. 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 a lot of associated software implementations, both commercial (e.g., in MATLAB[1] [21, 20]), copyrighted freeware (e.g., in the SLICOT Library [5, 28]), or in the public domain (e.g., in Scilab [8]). The reliability, efficiency, and functionality of various solvers differ significantly from package to package.

This paper presents several solvers for linear matrix equations available in the SLICOT Library (**S**ubroutine **L**ibrary **I**n **CO**ntrol **T**heory), that 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 [1], BLAS [9, 10, 19], and their counterparts for distributed memory computers, e.g., ScaLAPACK [6] and PBLAS, this library enables to exploit the potential of modern high-performance computer architectures.

The paper also presents some performance improvements (concerning efficiency, reliability, and accuracy) offered by the SLICOT tools, in comparison with equivalent computations performed by the MATLAB functions included in the MATLAB Control System Toolbox up to Release 13 of MATLAB. The results show that, at comparable or better accuracy, SLICOT computations are several times faster than MATLAB computations; moreover, the underlying problem structure is often better exploited. Note that from Release 14, MATLAB's linear matrix equation solvers will be based on the SLICOT codes presented in this paper. Therefore this paper can also be seen as a performance analysis of the improved Control Toolbox functions. Still, SLICOT routines provide a larger functionality with respect to the generality of equations that can be solved and the evaluation of sensitivity and errors in the computed solutions than what has been included in the MATLAB toolbox. We will highlight these features in the next two sections. In Section 4, we describe the examples used for testing the linear matrix equation solvers. The extensive evaluation of the numerical performance of the SLICOT linear equation solvers, given in Section 5, constitutes the major part of this paper.

2 Sylvester and Lyapunov Equations

Sylvester and Lyapunov equations are linear matrix equations. In a general setting, these equations can be defined as follows, where the notation $\mathrm{op}(M)$ denotes either the matrix M, or its transpose, M^T, A, B, $\mathrm{op}(D)$, 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.

- Continuous-time and discrete-time Sylvester equations:

$$\begin{aligned} \mathrm{op}(A)\, X \pm X\, \mathrm{op}(B) &= \sigma C; & (1)\\ \mathrm{op}(A)\, X\, \mathrm{op}(B) \pm X &= \sigma C; & (2) \end{aligned}$$

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

- Continuous-time and discrete-time[2] Lyapunov equations:

$$\begin{aligned} \mathrm{op}(A)^T X + X\,\mathrm{op}(A) &= \sigma C; & (3)\\ \mathrm{op}(A)^T X\,\mathrm{op}(A) - X &= \sigma C; & (4) \end{aligned}$$

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

$$\begin{aligned} \mathrm{op}(A)^T X + X\,\mathrm{op}(A) &= -\sigma^2\,\mathrm{op}(D)^T\,\mathrm{op}(D)\,; & (5)\\ \mathrm{op}(A)^T X\,\mathrm{op}(A) - X &= -\sigma^2\,\mathrm{op}(D)^T\,\mathrm{op}(D)\,; & (6) \end{aligned}$$

- Generalized Sylvester equation:

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

  or the "transposed" equation

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

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

$$\begin{aligned} \mathrm{op}(A)^T X\,\mathrm{op}(E) + \mathrm{op}(E)^T X\,\mathrm{op}(A) &= \sigma C; & (9)\\ \mathrm{op}(A)^T X\,\mathrm{op}(A) - \mathrm{op}(E)^T X\,\mathrm{op}(E) &= \sigma C; & (10) \end{aligned}$$

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

$$\begin{aligned} \mathrm{op}(A)^T X\,\mathrm{op}(E) + \mathrm{op}(E)^T X\,\mathrm{op}(A) &= -\sigma^2\,\mathrm{op}(D)^T\,\mathrm{op}(D)\,; & (11)\\ \mathrm{op}(A)^T X\,\mathrm{op}(A) - \mathrm{op}(E)^T X\,\mathrm{op}(E) &= -\sigma^2\,\mathrm{op}(D)^T\,\mathrm{op}(D)\,. & (12) \end{aligned}$$

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., [25, 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) [4] for Sylvester equations (for A, B general, or in RSF), or Lyapunov equations (for A general, or in RSF), with the variant from [3] for the discrete-time case; the Hessenberg-Schur method in [13] for standard Sylvester equations, i.e., with $\mathrm{op}(M) = M$ (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 [14] of the Bartels–Stewart method for stable Lyapunov equations; and extensions of the above methods for generalized Sylvester [15] and Lyapunov equations [12], [24].

The ability to work with the $\mathrm{op}(\cdot)$ 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 $\mathrm{op}(A) = A^T$, while observability Gramians can be defined as solutions of stable Lyapunov equations with $\mathrm{op}(A) = A$. When both controllability and observability Gramians are needed (e.g., in model reduction

[2] the discrete-time Lyapunov equation is also called Stein equation

computations as mentioned in the Introduction), then the same real Schur form of A can be used by a solver able to cope with $\mathrm{op}(\cdot)$, and this significantly improves the efficiency.

The term "stable" means that all eigenvalues of the matrix A (or of the matrix pencil $A - \lambda E$, for generalized stable Lyapunov equations) must have negative real parts, in the continuous-time case, or moduli less than one, in the discrete-time case. The solvers for stable Lyapunov equations compute the Cholesky factor U of the solution matrix X, i.e., $X = \mathrm{op}(U)^T \mathrm{op}(U)$, directly. 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 $\mathrm{op}(D)^T \mathrm{op}(D)$ 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*. For instance, besides solving continuous-time or discrete-time Lyapunov equations, it is advisable to compute the *separation* of the matrices A and $-A^T$, or of A and A^T, respectively. The separation measures the sensitivity of the equation to perturbations in the data, and it is defined by

$$\mathrm{sep}(A^T, -A) = \min_{Z \neq 0} \frac{\|A^T Z + ZA\|}{\|Z\|} = \sigma_{\min}(P), \tag{13}$$

$$\mathrm{sepd}(A^T, A) = \min_{Z \neq 0} \frac{\|A^T ZA - Z\|}{\|Z\|} = \sigma_{\min}(P), \tag{14}$$

in the continuous-time or discrete-time case, respectively, where $\sigma_{\min}(P)$ is the minimal singular value of the matrix P, and

$$P = I_n \otimes A^T + A^T \otimes I_n, \tag{15}$$

$$P = A^T \otimes A^T - I_{n^2}, \tag{16}$$

respectively; the symbol $\otimes$ stands for the Kronecker product of two matrices, $X \otimes Y = (x_{ij} Y)$. Estimates of the separation quantities can be computed very efficiently.

The corresponding sensitivity measure for Sylvester equations is the separation of the matrices A and B, defined similarly as above. For generalized Sylvester equations one can optionally compute a Dif estimate, $\mathrm{Dif}[(A, E), (B, F)]$, which measures the separation of the spectrum of the matrix pair (A, E) from the spectrum of the matrix pair (B, F) [15].

Such measures, as well as condition number estimates and forward error bounds are returned by several routines of the SLICOT Library [26]. This allows to judge the reliability of the computed solution while the only way to obtain an error measure in other control packages is to compute the residual which can be misleading if the condition of the problem is big. We consider this as a significant advantage in reliability and functionality of the software available in SLICOT.

3 Solvers for Sylvester and Lyapunov Equations

Solvers for Sylvester and Lyapunov equations are available in all major control systems software packages. The specific high-level interfaces of these solvers in MATLAB and SLICOT are briefly described in the following paragraphs.

The MATLAB (Release 13) Control System Toolbox includes two solvers for continuous-time Lyapunov equations, one solver for discrete-time Lyapunov equations, and two solvers for continuous-time Sylvester equations, namely:

`lyap` / `lyap2`	Solve continuous-time Lyapunov equations, if two arguments are given,
`dlyap`	Solve discrete-time Lyapunov equations,
`lyap` / `lyap2`	Solve continuous-time Sylvester equations, if three arguments are given.

Their usage is described in the following. The commands

```
X = lyap(A, C);
X = dlyap(A, C);
```

compute the unique solution X of the continuous-time Lyapunov equation or of the Stein equation

$$AX + XA^T = -C, \quad (17)$$

$$AXA^T - X = -C. \quad (18)$$

Unfortunately, a closer look at the equations (3) and (17) or (4) and (18) reveals that the right hand side of the equation (17) and (18) differs in the sign as well as the transposed matrix A is used in a different way. Note that the MATLAB command `lyap2` is used in the same way.

The command

```
X = lyap(A, B, C);
```

computes the unique solution X of the continuous-time Sylvester equation

$$AX + XB = -C. \quad (19)$$

The Sylvester equation (19) has again a different right hand side. There are no solvers for discrete-time Sylvester equations, generalized Sylvester- and Lyapunov equations in MATLAB releases prior to Release 14, and the existing solvers in Release 14 are based on SLICOT routines.

To calculate the solution of the continuous-time Lyapunov equation or Sylvester equation, `lyap` first performs the real Schur decomposition and converts it afterwards to the complex form, while `lyap2` performs only an eigenvalue decomposition. The discrete-time Lyapunov equation is reduced to the continuous one.

The SLICOT Library contains 16 "user-callable" Fortran 77 routines for Sylvester and Lyapunov equations. There are routines computing estimates for condition numbers, and forward error bounds for Lyapunov equations, which enable to assess the accuracy of the results and the sensitivity of the equations to perturbations in the data. The library also includes several additional, "programmer-callable" routines. Detailed documentation of all these routines is available as HTML files at the SLICOT ftp site accessible from the SLICOT hyperlink of the NICONET Web page

`http://www.win.tue.nl/niconet`

While the use of Fortran routines is more difficult, compared with user-friendly environments, like MATLAB, it enables to significantly increase the computational efficiency.

In order to enhance the user-friendliness of the efficient and reliable SLICOT Fortran routines, MATLAB or Scilab interfaces are provided for common control system analysis and design calculations, as shown below for Sylvester and Lyapunov equations. Two MEX-file implementations have been designed, `linmeq`, for standard linear matrix equations, and `genleq`, for generalized linear matrix equations. These

MEX-files call all SLICOT routines needed to perform the required task. The selection of the appropriate problem and solver is based on option parameters. For users' convenience, MATLAB functions are provided for each problem class. These MATLAB functions call the associated MEX-file. Executable MEX-files are provided on the SLICOT ftp site for PC platforms under Windows 95/98/00/ME/NT, or for Sun Solaris platforms (with Fortran 95). Linux versions are under current development.

The following MATLAB functions for Sylvester and Lyapunov-like equations are available:

`slsylv`	Solve continuous-time Sylvester equations.
`sldsyl`	Solve discrete-time Sylvester equations.
`sllyap`	Solve continuous-time Lyapunov equations.
`slstei`	Solve Stein equations.
`slstly`	Solve stable continuous-time Lyapunov equations.
`slstst`	Solve stable Stein equations.
`slgesg`	Solve generalized Sylvester equations.
`slgely`	Solve generalized continuous-time Lyapunov equations.
`slgest`	Solve generalized Stein equations.
`slgsly`	Solve stable generalized continuous-time Lyapunov equations.
`slgsst`	Solve stable generalized Stein equations.

Details on the use of these MATLAB functions are given in the sequel. The commands

```
X = slsylv(A, B, C, flag, trans, Schur);
X = sldsyl(A, B, C, flag, trans, Schur);
```

compute the unique solution X of a continuous-time Sylvester equation (1), or of a discrete-time Sylvester equation (2), respectively. The optional input parameter `flag` is a vector of length 2, which specifies the structure of A and/or B. The elements `flag(1)` and `flag(2)` refer to A and B, respectively. An input matrix is assumed to be quasi-upper triangular (or in real Schur form) if the corresponding element of `flag` is 1, and an input matrix is assumed to be upper Hessenberg if the corresponding element of `flag` is 2; otherwise, that matrix is a general matrix (default). The optional parameter `trans` specifies the operator $\mathrm{op}(\cdot)$ for the matrices A and B, as follows

$$
\begin{array}{lclll}
\texttt{trans} & = & 0: & \mathrm{op}(A) = A; & \mathrm{op}(B) = B \quad \text{(default)}; \\
\texttt{trans} & = & 1: & \mathrm{op}(A) = A^T; & \mathrm{op}(B) = B^T; \\
\texttt{trans} & = & 2: & \mathrm{op}(A) = A^T; & \mathrm{op}(B) = B; \\
\texttt{trans} & = & 3: & \mathrm{op}(A) = A; & \mathrm{op}(B) = B^T.
\end{array}
\tag{20}
$$

The optional parameter `Schur` specifies the method to be used for the solution, as follows. If `Schur` = 1, the Hessenberg-Schur method is used by the solver, that is, one matrix is reduced to Hessenberg form, and the other matrix is reduced to Schur form (default); if `Schur` = 2, the Schur method is used, that is, both matrices are reduced to their Schur forms. If one or both matrices are already reduced to Schur/Hessenberg forms, this can be specified by `flag(1)` and `flag(2)`. For general matrices, the Hessenberg-Schur method is significantly more efficient than the Schur method.

The commands

```
[X, sep] = sllyap(A, C, flag, trans);
[X, sepd] = slstei(A, C, flag, trans);
```

compute the unique symmetric solution X of a continuous-time Lyapunov equation (3) and of a discrete-time Lyapunov equation (4), respectively. If `flag` = 1, then A is assumed to be quasi upper triangular (or in RSF); otherwise, A is a general matrix (default). If `trans` = 0, then $\mathrm{op}(A) = A$ (default); otherwise,

$\text{op}(A) = A^T$. The optional output parameter sep or sepd returns an estimate of the separation of the matrices A and $-A^T$, defined by (13), or of the separation of the matrices A and A^T, defined by (14), respectively.

The following two commands compute the Cholesky factor U of the unique symmetric positive semi-definite solution, $\text{op}(U)^T \text{op}(U)$, of a stable continuous-time Lyapunov equation (5), or a stable discrete-time Lyapunov equation (6), respectively,

```
U = slstly(A, B, flag, trans);
U = slstst(A, B, flag, trans);
```

where flag and trans are the optional parameters defined above for Lyapunov equations.

The command

```
[X, Y, dif] = slgesg(A, E, B, F, G, H, flag, trans);
```

computes the unique solutions (X, Y) of the generalized linear matrix equation pairs (7), if trans $= 0$ (default), or the "transposed" equation pairs (8), if trans $\neq 0$. The optional input parameter flag is a vector with two elements, characterizing the structure of the matrix pairs. Specifically, flag(1) and flag(2) refer to the matrix pair (A, E) and (B, F), respectively. If flag(i) $= 1$, the matrix pair i is assumed to be in a generalized Schur form; otherwise, that pair is in a general form. Default value is flag = [0,0], that is, both pairs (A, E), and (B, F) are in general forms. The optional output parameter dif returns an estimate of the quantity $\text{dif}[(A, E), (B, F)]$, which generalizes the notion of separation of two matrices.

The commands

```
[X, sep] = slgely(A, E, C, flag, trans);
[X, sep] = slgest(A, E, C, flag, trans);
```

compute the unique symmetric solution X of a generalized continuous-time Lyapunov equation (9), and a generalized Stein (discrete-time Lyapunov) equation (10), respectively. If flag $= 1$, it is assumed that (A, E) is in generalized Schur form; otherwise, (A, E) is in general form (default). The optional output parameter sep returns the separation, $\text{sep}(A, E)$.

Similarly, the following two commands compute a Cholesky factor U of the unique symmetric positive semi-definite solution $\text{op}(U)^T \text{op}(U)$ of a stable generalized continuous-time Lyapunov equation (11), or of a stable generalized discrete-time Lyapunov equation (12), respectively,

```
U = slgsly(A, E, D, flag, trans);
U = slgsst(A, E, D, flag, trans);
```

Remark. From Release 14, MATLAB uses the SLICOT codes for solving (generalized) Lyapunov, (generalized) Stein, and Sylvester equations. Included are also the SLICOT solvers for the stable Lyapunov and Stein equation, but not the generalized Sylvester equation solver. Unfortunately, the sep and dif calculations are also not included, so that the conditioning of the linear matrix equations is not accessible via the new Control Toolbox functions.

4 Numerical examples

This subsection describes in brief the examples which were used to test and compare speed and accuracy of the solvers for

- discrete-time and continuous-time Lyapunov equations,
- stable discrete-time and stable continuous-time Lyapunov equations, where the Cholesky factor U of the solution is computed,
- discrete-time and continuous-time Sylvester equations,
- generalized discrete-time and continuous-time Lyapunov equations,
- generalized stable discrete-time and continuous-time Lyapunov equations, where the Cholesky factor U of the solution is computed,
- generalized Sylvester equations,

implemented in MATLAB and SLICOT. In order to get an example of a Lyapunov equation with known solution, we start in a first step with the auxiliary equations

$$\begin{aligned} A_0^T X_0 + X_0 A_0 &= -C_0^T C_0, & (21)\\ A_0^T X_0 A_0 - X_0 &= -C_0^T C_0, & (22)\end{aligned}$$

where A_0 is a diagonal matrix. C_0 can be a suitable row vector or a diagonal matrix. Because of this choice, the entries of the solutions can be computed easily as:

$$(X_0)_{ij} = -\frac{(C_0)_i (C_0)_j}{(A_0)_{ii} + (A_0)_{jj}} \tag{23}$$

for the continuous-time Lyapunov equation (21) and as

$$(X_0)_{ij} = -\frac{(C_0)_i (C_0)_j}{(A_0)_{ii} (A_0)_{jj} - 1} \tag{24}$$

for the discrete-time Lyapunov equation (22). In addition we choose another example of continuous-time and discrete-time Lyapunov equations, where the matrix A_0 of (21), (22) is a Jordan block matrix in the following manner:

$$J = \begin{bmatrix} \lambda & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & \lambda \end{bmatrix}. \tag{25}$$

This matrix has only one real (multiple) eigenvalue, assuming $\lambda \in \mathbb{R}$. Again, the entries of the solution can be computed recursively as:

$$(X_0)_{ij} = \begin{cases} -\dfrac{(C_0)_1 (C_0)_1}{2\lambda}, & \text{if} \quad i = 1 \quad \text{and} \quad j = 1 \\[2ex] -\dfrac{(C_0)_1 (C_0)_j + (X_0)_{1,j-1}}{2\lambda}, & \text{if} \quad i = 1 \quad \text{and} \quad j \in \{2, \ldots, n\} \\[2ex] -\dfrac{(C_0)_i (C_0)_1 + (X_0)_{i-1,1}}{2\lambda}, & \text{if} \quad j = 1 \quad \text{and} \quad i \in \{2, \ldots, n\} \\[2ex] -\dfrac{(C_0)_i (C_0)_j + (X_0)_{i-1,j} + (X_0)_{i,j-1}}{2\lambda}, & \text{if} \quad i, j \in \{2, \ldots, n\} \end{cases} \tag{26}$$

for continuous-time Lyapunov equations and

$$
(X_0)_{ij} = \begin{cases} -\dfrac{(C_0)_1(C_0)_1}{\lambda^2-1}, & \text{if} \quad i=1 \quad \text{and} \quad j=1 \\[2ex] -\dfrac{(C_0)_1(C_0)_j + \lambda\ (X_0)_{1,j-1}}{\lambda^2-1}, & \text{if} \quad i=1 \quad \text{and} \quad j\in\{2,\ldots,n\} \\[2ex] -\dfrac{(C_0)_i(C_0)_1 + \lambda\ (X_0)_{i-1,1}}{\lambda^2-1}, & \text{if} \quad j=1 \quad \text{and} \quad i\in\{2,\ldots,n\} \\[2ex] -\dfrac{(C_0)_i(C_0)_j + \lambda\ \big((X_0)_{i-1,j} + (X_0)_{i,j-1}\big)}{\lambda^2-1}, & \text{if} \quad i,j\in\{2,\ldots,n\} \end{cases} \tag{27}
$$

for discrete-time Lyapunov equations.

In the same way, we construct examples with known solution for Sylvester equations

$$
\begin{aligned} A_0X_0 + X_0B_0 &= -C_0 && (28)\\ A_0X_0B_0 + X_0 &= -C_0, && (29) \end{aligned}
$$

where A_0, B_0 are appropriate matrices. The entries of the solutions are given by

$$
(X_0)_{ij} = -\frac{(C_0)_{ij}}{(A_0)_{ii} + (B_0)_{jj}} \tag{30}
$$

for the continuous-time Sylvester equation (28) and by

$$
(X_0)_{ij} = -\frac{(C_0)_{ij}}{(A_0)_{ii}(B_0)_{jj} + 1} \tag{31}
$$

for the discrete-time Sylvester equation (29).

In a second step we construct a transformation matrix $T \in \mathbb{R}^{n\times n}$ as

$$
T = H_2 S H_1 \tag{32}
$$

where

$$
\begin{aligned} H_1 &= I_n - \frac{2}{n}\, ee^T, && e = [1,1,\ldots,1]^T, \\ H_2 &= I_n - \frac{2}{n}\, ff^T, && f = [1,-1,\ldots,(-1)^{n-1}]^T, \\ S &= \operatorname{diag}(1,s,\ldots,s^{n-1}), && s>1. \end{aligned}
$$

Using different values of the scalar s, it is possible to change the condition of T. The transformed matrices A, B, C, X are given by

$$
A = TA_0T^{-1}, \quad C = C_0T^{-1}, \quad X = T^{-T}X_0T^{-1}
$$

for the Lyapunov equations and by

$$
A = T^{-T}A_0T^T, \quad B = TB_0T^{-1}, \quad C = T^{-T}C_0T^{-1}, \quad X = T^{-T}X_0T^{-1}
$$

for the Sylvester equations. These transformations can be computed easily with high precision.

Example 1. For the continuous-time Lyapunov equation choose

$$
\begin{aligned} A_0 &= \operatorname{diag}(-1,-a,-a^2,\ldots,-a^{n-1}), && a>1 \\ C_0 &= [\,1,2,\ldots,n\,], \end{aligned}
$$

where the parameter a regulates the distribution of the eigenvalues of A_0.

Example 2. For the continuous-time Lyapunov equation choose now

$$\begin{aligned} A_0 &= J, \qquad \lambda < 0 \\ C_0 &= [1 \quad 0 \quad \dots \quad 0], \end{aligned}$$

where J is a Jordan block matrix with one real eigenvalue λ.

Example 3. For the discrete-time Lyapunov equation choose

$$\begin{aligned} A_0 &= \operatorname{diag}(0, \frac{a-1}{a+1}, \frac{a^2-1}{a^2+1}, \dots, \frac{a^{n-1}-1}{a^{n-1}+1}), \qquad a > 1 \\ C_0 &= [1, 2, \dots, n], \end{aligned}$$

where the parameter a regulates the distribution of the eigenvalues of A_0 as well.

Example 4. For the discrete-time Lyapunov equation choose

$$\begin{aligned} A_0 &= J, \qquad -1 < \lambda \leqslant 0 \\ C_0 &= [1 \quad 0 \quad \dots \quad 0], \end{aligned}$$

where J is a Jordan block matrix with one real eigenvalue λ.

Examples 1 and 3 are implemented in the benchmark collections CTLEX [16] and DTLEX [17] provided by the SLICOT library. It has to be mentioned that even for small values of the parameters a and s and moderate dimension n the Lyapunov equations tend to be ill-conditioned. Furthermore, loosely speaking, the distribution of the eigenvalues of A in these examples is so wide, that it is difficult to find appropriate values for a and s avoiding a numerical indefinite matrix in order to calculate the Cholesky factor of the solution.

Example 5. For the stable continuous-time Lyapunov equation choose

$$\begin{aligned} A_0 &= \operatorname{diag}(-1, -a, -a^2, \dots, -a^{n-1}), \qquad a > 1 \\ C_0 &= \operatorname{diag}[1, 2, \dots, n], \end{aligned}$$

where the parameter a regulates the distribution of the eigenvalues of A_0.

Example 6. For the stable discrete-time Lyapunov equation choose

$$\begin{aligned} A_0 &= \operatorname{diag}(0, \frac{a-1}{a+1}, \frac{a^2-1}{a^2+1}, \dots, \frac{a^{n-1}-1}{a^{n-1}+1}), \qquad a > 1 \\ C_0 &= \operatorname{diag}[1, 2, \dots, n], \end{aligned}$$

where the parameter a regulates the distribution of the eigenvalues of A_0.

Note that in Examples 5 and 6 the solution X_0 is a diagonal matrix. Choosing small values for the parameter s the eigenvalues of the matrix X_0 are an approximation of the eigenvalues of the transformed matrix X.

Example 7. For the continuous-time Sylvester equation choose

$$\begin{aligned} A_0 &= \operatorname{diag}(-1, -a, -a^2, \dots, -a^{n-1}), \qquad a > 1 \\ B_0 &= \operatorname{diag}(-1, -b, -b^2, \dots, -b^{n-1}), \qquad b > 1 \\ C_0 &= [1, 2, \dots, n], \end{aligned}$$

where the parameters a and b regulate the distribution of the eigenvalues of A_0 and B_0, respectively.

Example 8. For the discrete-time Sylvester equation choose

$$\begin{aligned} A_0 &= \operatorname{diag}(0, \frac{a-1}{a+1}, \frac{a^2-1}{a^2+1}, \ldots, \frac{a^{n-1}-1}{a^{n-1}+1}), \qquad a > 1 \\ B_0 &= \operatorname{diag}(0, \frac{b-1}{b+1}, \frac{b^2-1}{b^2+1}, \ldots, \frac{b^{n-1}-1}{b^{n-1}+1}), \qquad b > 1 \\ C_0 &= [\,1, 2, \ldots, n\,], \end{aligned}$$

where the parameters a and b regulate the distribution of the eigenvalues of A_0 and B_0, respectively.

In order to get an example of a generalized Lyapunov equation where the exact solution matrix is known, we define the matrices A and E as

$$\begin{aligned} A &= (2^{-t}-1)I_n + \operatorname{diag}(1, 2, \ldots, n) + U^T \\ E &= I_n + 2^{-t}U \end{aligned}$$

with

$$U = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 1 & \cdots & 1 & 0 \end{bmatrix} \tag{33}$$

Increasing the value of the parameter t tends to lower the accuracy of the numerically computed solution matrices.

Example 9. For the generalized continuous-time Lyapunov equation choose the following solution

$$X = e\,e^T, \qquad e = [\,1 \quad 1 \quad \ldots \quad 1\,]^T,$$

and the corresponding right hand side is computed as

$$Y = A^T XE + E^T XA.$$

Example 10. For the generalized discrete-time Lyapunov equation choose the following solution

$$X = e\,e^T, \qquad e = [\,1 \quad 1 \quad \ldots \quad 1\,]^T,$$

and the corresponding right hand side is computed as

$$Y = A^T XA - E^T XE.$$

These last two examples are also implemented in the benchmark collections CTLEX [16] and DTLEX [17] provided by the SLICOT library. Unfortunately, the solution X of the last two examples is singular, so that it is not possible to compute the Cholesky factor $X = UU^T$.

For constructing an example for generalized stable Lyapunov equations and generalized Sylvester equations, we start in a first step with the auxiliary equations

$$A_0^T X_0 E_0 + E_0^T X_0 A_0 = -C_0^T C_0 \tag{34}$$

$$A_0^T X_0 A_0 - E_0^T X_0 E_0 = -C_0^T C_0, \tag{35}$$

or, respectively,

$$
\begin{aligned}
A_0 X_0 - Y_0 B_0 &= G_0 \\
E_0 X_0 - Y_0 F_0 &= H_0,
\end{aligned} \tag{36}
$$

where A_0, E_0 and B_0, F_0 are diagonal matrices. The entries of the solution can now be computed easily as

$$
(X_0)_{ij} = -\frac{(C_0)_i (C_0)_j}{(A_0)_{ii}(E_0)_{jj} + (E_0)_{ii}(A_0)_{jj}} \tag{37}
$$

for generalized stable continuous-time Lyapunov equations,

$$
(X_0)_{ij} = -\frac{(C_0)_i (C_0)_j}{(A_0)_{ii}(A_0)_{jj} - (E_0)_{ii}(E_0)_{jj}} \tag{38}
$$

for generalized stable discrete-time Lyapunov equations and

$$
\begin{aligned}
(X_0)_{ij} &= \frac{(F_0)_{jj}(G_0)_{ij} - (B_0)_{jj}(H_0)_{ij}}{(A_0)_{ii}(F_0)_{jj} - (E_0)_{ii}(B_0)_{jj}} \\
(Y_0)_{ij} &= \frac{(E_0)_{ii}}{(F_0)_{jj}}(X_0)_{ij} - \frac{1}{(F_0)_{jj}}(H_0)_{ij} = \frac{(E_0)_{ii}(G_0)_{ij} - (A_0)_{ii}(H_0)_{ij}}{(A_0)_{ii}(F_0)_{jj} - (E_0)_{ii}(B_0)_{jj}}
\end{aligned} \tag{39}
$$

for generalized Sylvester equations. Using the transformation matrix T, we construct in a second step the transformed matrices

$$
A = TA_0T^{-1}, \quad E = TE_0T^{-1}, \quad C = C_0T^{-1}, \quad X = T^{-T}X_0T^{-1}
$$

for generalized Lyapunov equations and

$$
\begin{gathered}
A = T^{-T}A_0T^T, \quad B = TB_0T^{-1}, \quad E = T^{-T}E_0T^T, \quad F = TF_0T^{-1}, \\
G = T^{-T}G_0T^{-1}, \quad H = T^{-T}H_0T^{-1}, \quad X = T^{-T}X_0T^{-1}, \quad Y = T^{-T}Y_0T^{-1}
\end{gathered}
$$

for generalized Sylvester equations.

Example 11. For the generalized stable continuous-time Lyapunov equation choose

$$
\begin{aligned}
A_0 &= \operatorname{diag}(-1, -a, -a^2, \ldots, -a^{n-1}), \qquad a > 1 \\
E_0 &= \operatorname{diag}(1, e, e^2, \ldots, e^{n-1}), \qquad e > 1 \\
C_0 &= \operatorname{diag}[1, 2, \ldots, n],
\end{aligned}
$$

where the parameters a and e regulate the distribution of the eigenvalues of A_0 and E_0.

Example 12. For the generalized stable discrete-time Lyapunov equation choose

$$
\begin{aligned}
A_0 &= \operatorname{diag}(0, \frac{a-1}{a+1}, \frac{a^2-1}{a^2+1}, \ldots, \frac{a^{n-1}-1}{a^{n-1}+1}), \qquad a > 1 \\
E_0 &= \operatorname{diag}(1, e, e^2, \ldots, e^{n-1}), \qquad e > 1 \\
C_0 &= \operatorname{diag}[1, 2, \ldots, n],
\end{aligned}
$$

where the parameters a and e regulate the distribution of the eigenvalues of A_0 and E_0.

Example 13. For the generalized Sylvester equation choose

$$
\begin{aligned}
A_0 &= \operatorname{diag}(1, a, a^2, \ldots, a^{n-1}), \qquad a > 1 \\
B_0 &= \operatorname{diag}(-1, -b, -b^2, \ldots, -b^{n-1}), \qquad b > 1 \\
E_0 &= \operatorname{diag}(-1, -e, -e^2, \ldots, -e^{n-1}), \qquad e > 1 \\
F_0 &= \operatorname{diag}(-1, -f, -f^2, \ldots, -f^{n-1}), \qquad f > 1 \\
G_0 &= -v^T v, \qquad H_0 = v^T v, \qquad v = [1, 2, \ldots, n],
\end{aligned}
$$

where the parameters a, b, e and f regulate the distribution of the eigenvalues of the matrices A_0, B_0, E_0 and F_0.

Note that in all these examples the distribution of the eigenvalues is related to the condition of the respective Lyapunov or Sylvester equations and increasingly ill-conditioned examples can therefore be generated using the parameters.

5 Numerical results

This subsection presents typical performance results for some components of the SLICOT Library, called via the associated gateways. The calculations have been done on a PC computer with an AMD processor at 1.8 GHz, with 1 Gb memory and the relative machine precision $\epsilon = 2.22 \times 10^{-16}$, using Intel Fortran Compiler V7.0, optimized BLAS provided by MATLAB, and MATLAB 6.5 (R13).

These results show that SLICOT routines often outperform MATLAB calculations. While the accuracy is comparable, sometimes better but also sometimes worse, the gain in efficiency by calling SLICOT routines can be significant. Note that the results have been obtained by timing in MATLAB the equivalent computations using the MATLAB commands

`t_begin = cputime;` *command*; `t_end = cputime - t_begin;`

Even better efficiency is to be expected by calling the SLICOT Fortran routines directly (not through gateways), and similar accuracy/efficiency improvements are possible for other SLICOT computations. Better results will be obtained if the working space of the SLICOT functions is increased.

Figure 1 shows the execution times for SLICOT function `sllyap` and MATLAB functions `lyap` and `lyap2` (left plot), and the speed-up factor of `sllyap` versus `lyap` (right plot), for solving Lyapunov equations generated with Example 1 using the following parameters:

$$a = 1.005, \quad s = 1.005, \quad n = 5 : 5 : 1000.$$

In order to calculate the RSF of A, the SLICOT function `systra` is used. These two plots show that the SLICOT solver `sllyap` outperforms the MATLAB solver `lyap`. So between the dimension $n = 100$ and $n = 300$ SLICOT is up to 2.3 times as fast as MATLAB while for higher dimensions, starting at $n = 500$ SLICOT is still 1.5 times faster than MATLAB. In the case that A is in RSF, the SLICOT solver `sllyap` outperforms clearly the MATLAB solver `lyap`. This is also shown in Figure 1 (right plot), where the speed-up factor varies between 7.5 and 19.8. On the other hand, a closer look at the execution time of `sllyap` and `lyap2` reveals that these two solvers are comparable. To obtain a clear representation, Figure 1 (right plot) does not contain the comparison of SLICOT versus MATLAB's `lyap2`. The two peaks at $n = 640$ and $n = 960$ are caused by cache effects.

Although the accuracy of the MATLAB function `lyap2`, shown in Figure 2 (left plot), is worse in comparison with `lyap` and with the SLICOT function `sllyap`, the difference is marginal. Because of the fact that the accuracy is nearly identical if A is given in RSF, only the result of the SLICOT solver is shown for this set-up. It has to be mentioned that the condition varies between 1 to 60 for these parameters. In Figure 2 (right plot) the relative error comparison versus the condition is shown using Example 1 with the following parameters:

$$a = 1.02 \ldots 4.33, \quad s = 1.02 \ldots 4.33, \quad n = 10.$$

The difference between the SLICOT solver and the MATLAB solvers is marginal.

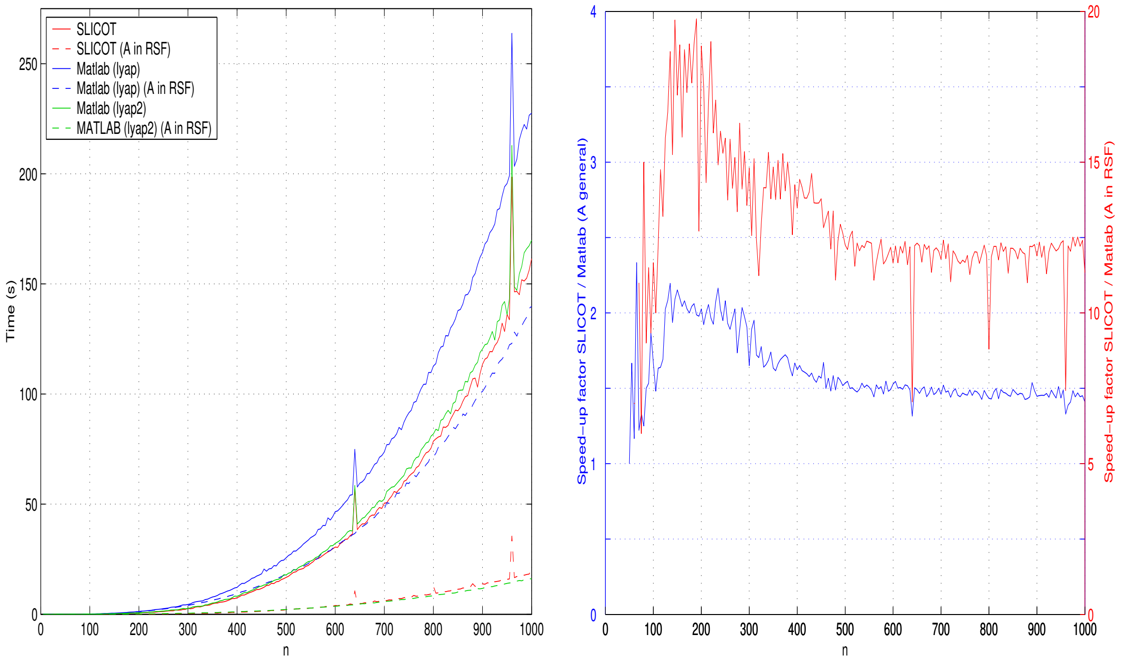


Figure 1: SLICOT **sllyap** versus MATLAB **lyap** and **lyap2** for generated Lyapunov equations using Example 1 with the parameters $a = 1.005$, $s = 1.005$. Timing comparison (left) and speed-up factor (right). In order to calculate the RSF of A, the SLICOT function **systra** is used.

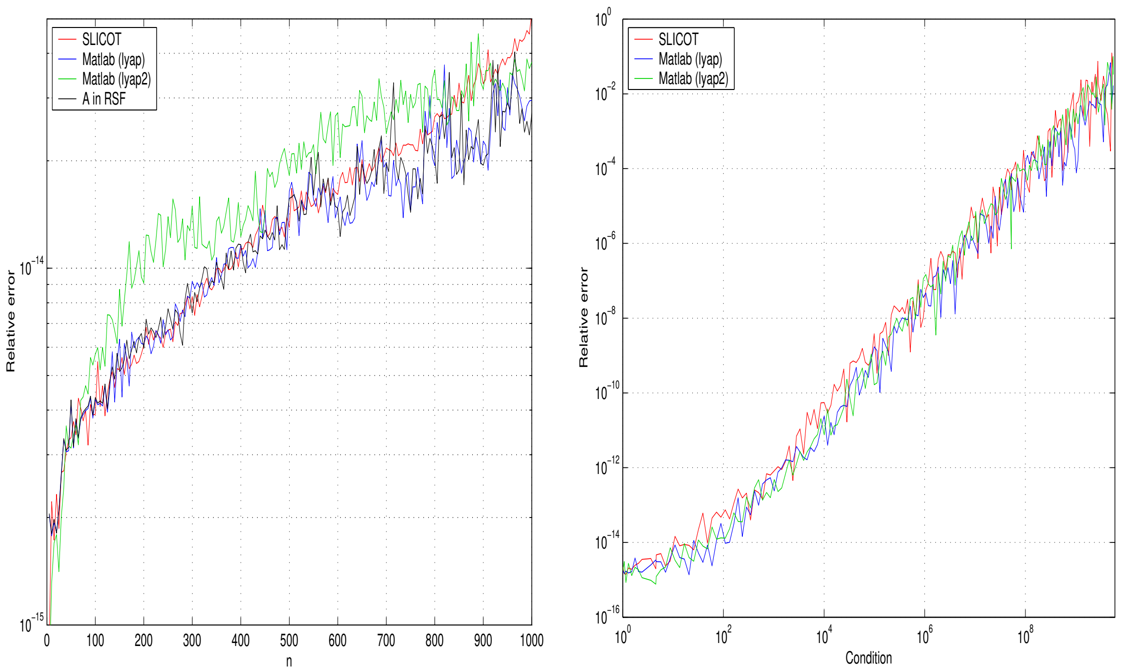


Figure 2: SLICOT **sllyap** versus MATLAB **lyap** and **lyap2** for generated Lyapunov equations using Example 1. The left plot shows the relative error comparison versus the dimension using the parameters $a = 1.005$ and $s = 1.005$ while the right plot reveals the relative error versus the condition. For the latter, the following parameters are chosen: $a = 1.02 \ldots 4.33$, $s = 1.02 \ldots 4.33$ and $n = 10$.

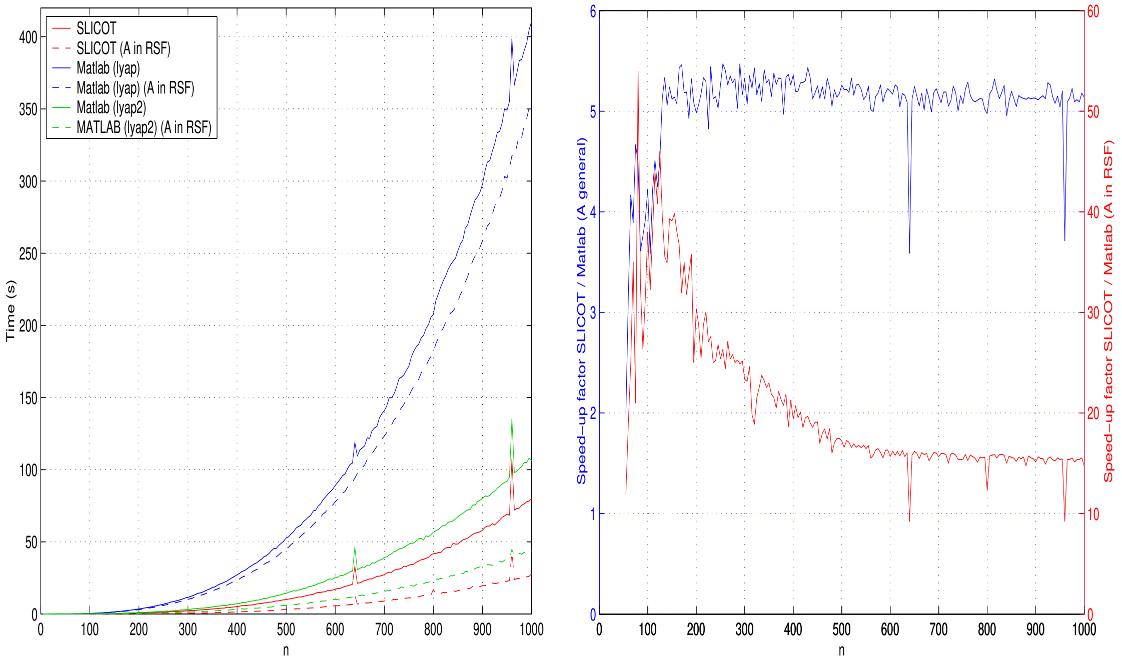


Figure 3: SLICOT `sllyap` versus MATLAB `lyap` and `lyap2` for generated Lyapunov equations using Example 2 with the parameters $\lambda = -1.5$, $s = 1.01$. Timing comparison (left) and speed-up factor (right). In order to calculate the RSF of A, the SLICOT function `systra` is used.

Figure 3 shows the execution times for SLICOT function `sllyap` and MATLAB functions `lyap` and `lyap2` (left plot), and the speed-up factor of `sllyap` versus `lyap` (right plot), for solving Lyapunov equations generated with Example 2 using the following parameters:

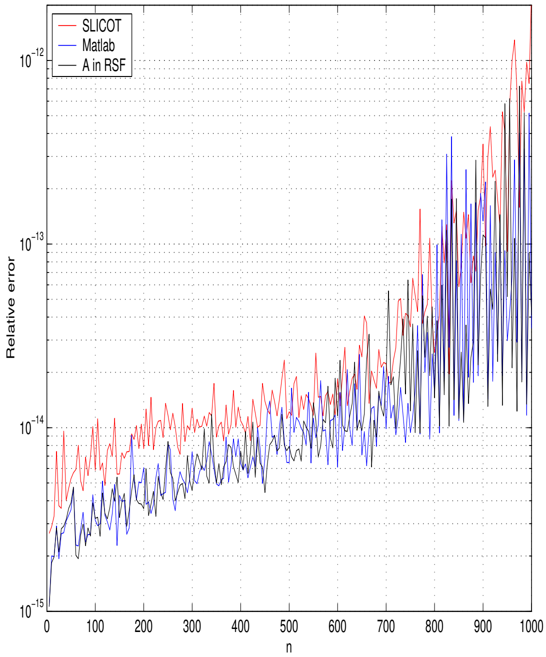
$$\lambda = -1.5, \quad s = 1.01, \quad n = 5:5:1000,$$

whereat the general matrix A and its real Schur decomposition via the SLICOT function `systra` is used. Hence, the SLICOT solver is much faster than the MATLAB solver `lyap` (up to 5 times as fast for general A and up to 45 times as fast for A in RSF), while the accuracy, shown in Figure 4 (left plot), is comparable. Indeed, the MATLAB solver `lyap2` is only about 25 % slower than the SLICOT solver, but `lyap2` is not useful to solve the Lyapunov equations generated with Example 2 because its relative error is higher than 1. It is important to mention that the MATLAB solver `lyap2` will not work if the matrix A of the Lyapunov equation contains repeated roots. Due to the fact that the accuracy of the SLICOT solver `sllyap` and the MATLAB solver `lyap` is nearly the same, if A is given in RSF, the result of `sllyap` is shown only. Note that the condition of the example with these chosen parameters varies between 1 and 1.5×10^{6}. In Figure 4 (right plot) the relative error comparison versus the condition is shown using Example 2 with the following parameters:

$$\lambda = -2.00 \ldots -0.1, \quad s = 1.5, \quad n = 10.$$

The difference between `sllyap` and `lyap` is marginal, if the condition varies between 1 and 10^{6}. But SLICOT gains up to five digits in accuracy, if the condition is further increased.

The efficiency of the SLICOT discrete-time Lyapunov solver is similar, as shown in Figure 5 and Figure 7. These figures reveal the execution times for SLICOT function `slstei` and MATLAB functions



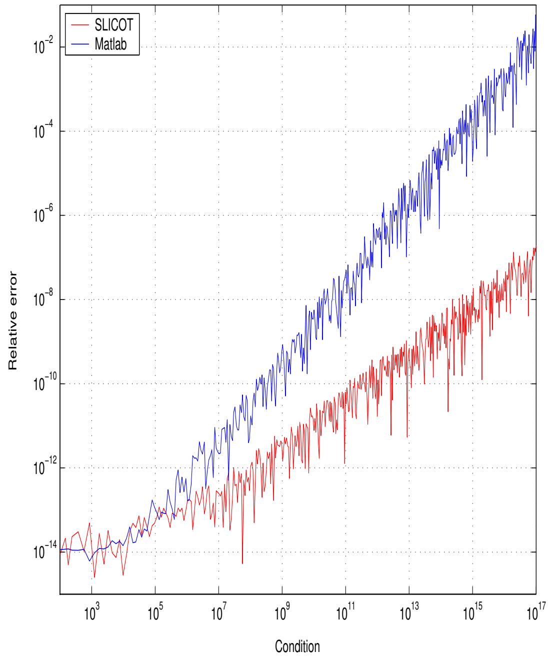


Figure 4: SLICOT **sllyap** versus MATLAB **lyap** for generated Lyapunov equations using Example 2. The left plot shows the relative error comparison versus the dimension using the parameters $\lambda = -1.5$ and $s = 1.01$ while the right plot reveals the relative error versus the condition. To achieve this we choose the following parameters: $\lambda = -2.00 \ldots -0.1$, $s = 1.5$ and $n = 10$.

`dlyap` (left plot), and the speed-up factor of `slstei` versus `dlyap` (right plot), for solving Stein equations generated with Example 3 using the following parameters:

$$a = 1.005, \quad s = 1.005, \quad n = 5 : 5 : 1000$$

(Figure 5) or generated with Example 4 using the parameters:

$$\lambda = -0.01, \quad s = 1.005, \quad n = 5 : 5 : 1000$$

(Figure 7). Using these parameters, the condition in Example 3 varies between 1 and 1.0×10^5 while in Example 4 it varies between 5 and 1.0×10^8.

Again, the solvers were tested with the general matrix A and with its real Schur decomposition computed using the SLICOT function `systra`. It is remarkable that for Example 4 the speed-up factor increases until the dimension $n = 200$ from 2 to 5.5 and stays at this level for higher dimensions. If A is in RSF, the speed-up factor even varies between 13 and 43. While for Example 3 the SLICOT solver `slstei` is between 1.3 and 2.4 times as fast as the the MATLAB solver `dlyap`. But if the matrix A is given in RSF the speed-up factor varies between 7 and 14. The SLICOT solver `slstei` loses one digit in Example 3, see Figure 6 (left plot), whereas using Example 3 with the parameters:

$$a = 1.005, \ldots 3.8 \quad s = 1.005 \ldots 3.8, \quad n = 10$$

`slstei` gains four digits, see Figure 6 (right plot). Employing Example 4, the accuracy of SLICOT solver `slstei` versus MATLAB solver `dlyap` is comparable and sometimes better while the condition for these parameters varies between 10^1 and 10^8, see Figure 8 (left plot). Using Example 4 with the parameters:

$$\lambda = -0.9 \ldots 0, \quad s = 1.2, \quad n = 10$$

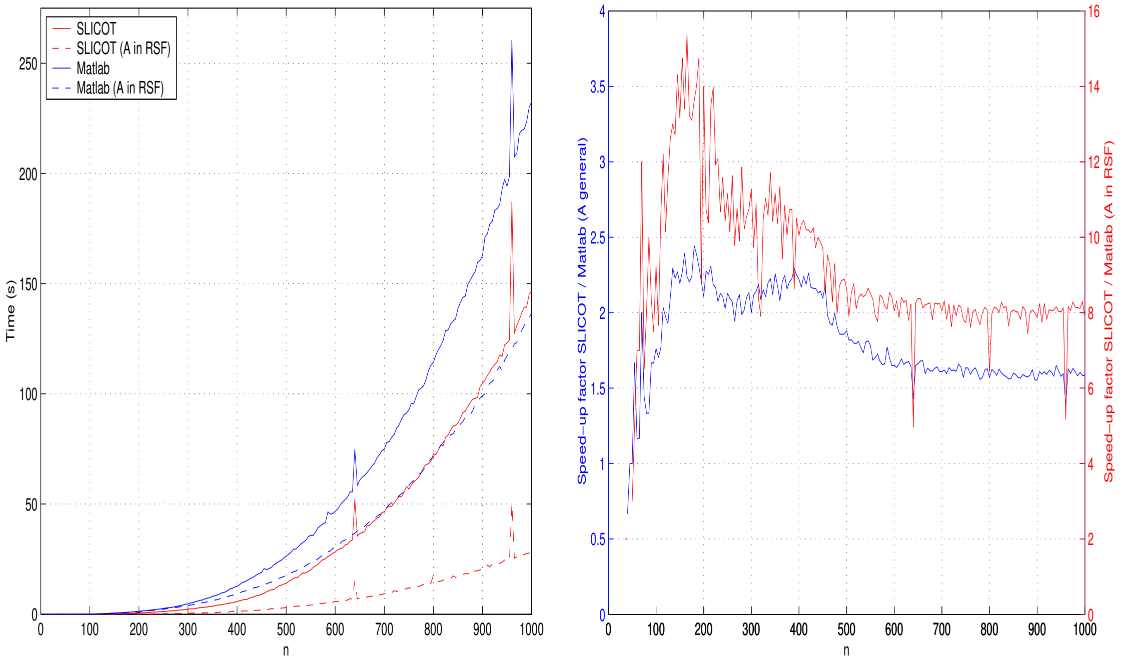


Figure 5: SLICOT **slstei** versus MATLAB **dlyap** for generated Stein equations using Example 3 with the parameters $a = 1.005$, $s = 1.005$. Timing comparison (left) and speed-up factor (right).

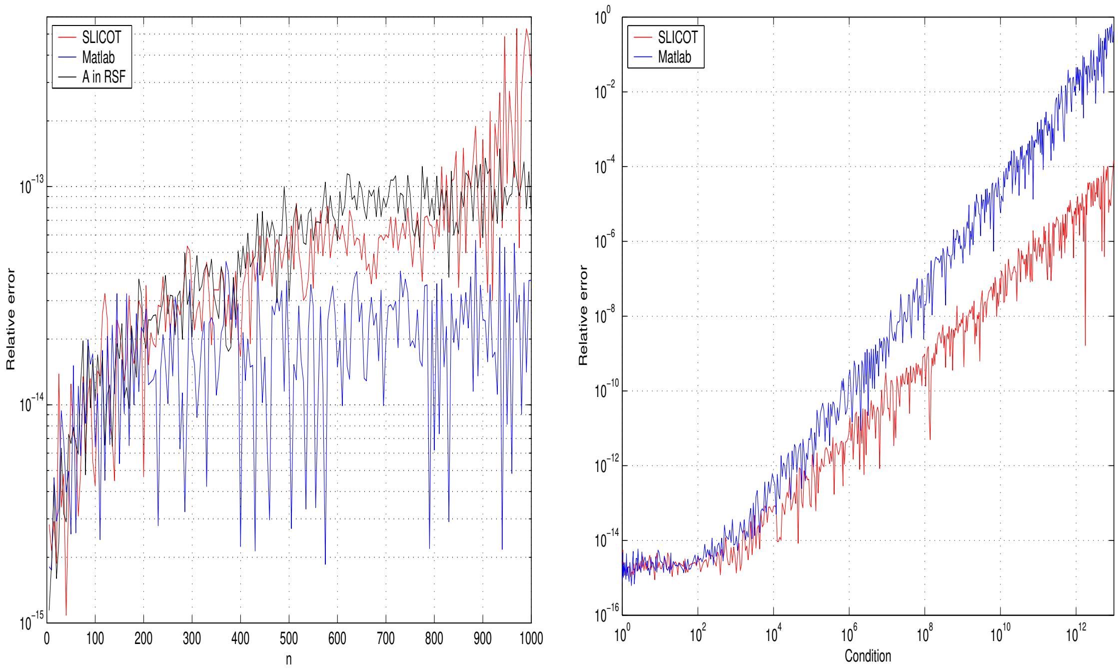


Figure 6: SLICOT **slstei** versus MATLAB **dlyap** for generated Stein equations using Example 3. The left plot shows the relative error comparison versus the dimension using the parameters $a = 1.005$ and $s = 1.005$ while the right plot reveals the relative error versus the condition. For the latter, the following parameters are chosen: $a = 1.005 \ldots 3.8$, $s = 1.005 \ldots 3.8$ and $n = 10$.

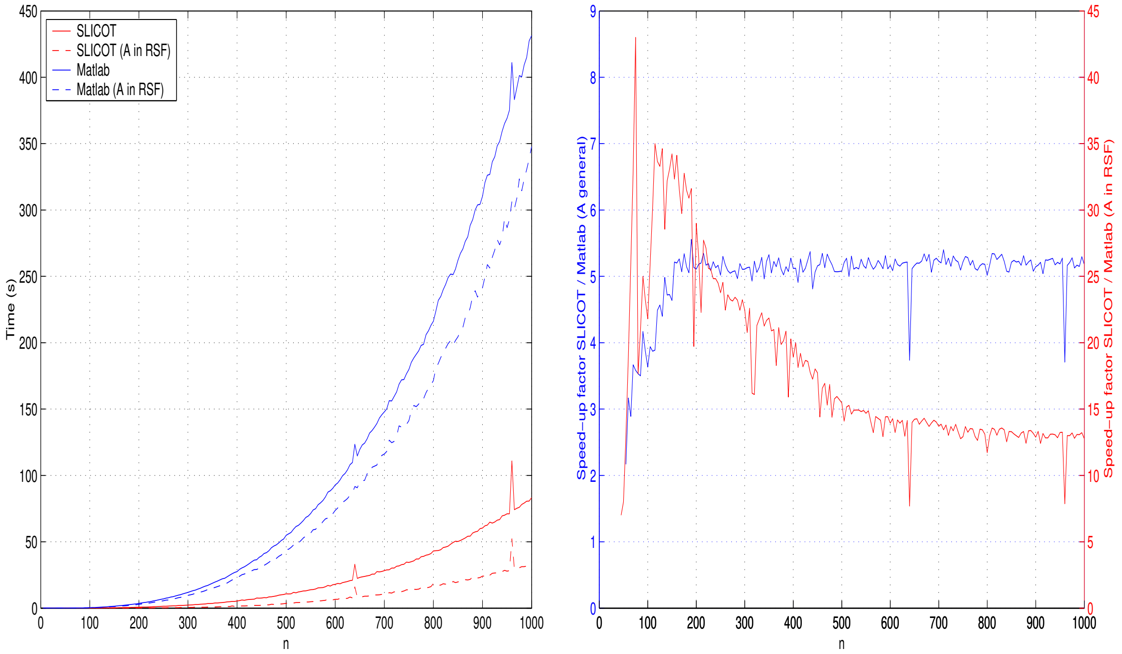


Figure 7: SLICOT **slstei** versus MATLAB **dlyap** for generated Stein equations using Example 4 with the parameters $\lambda = -0.01$, $s = 1.005$. Timing comparison (left) and speed-up factor (right).

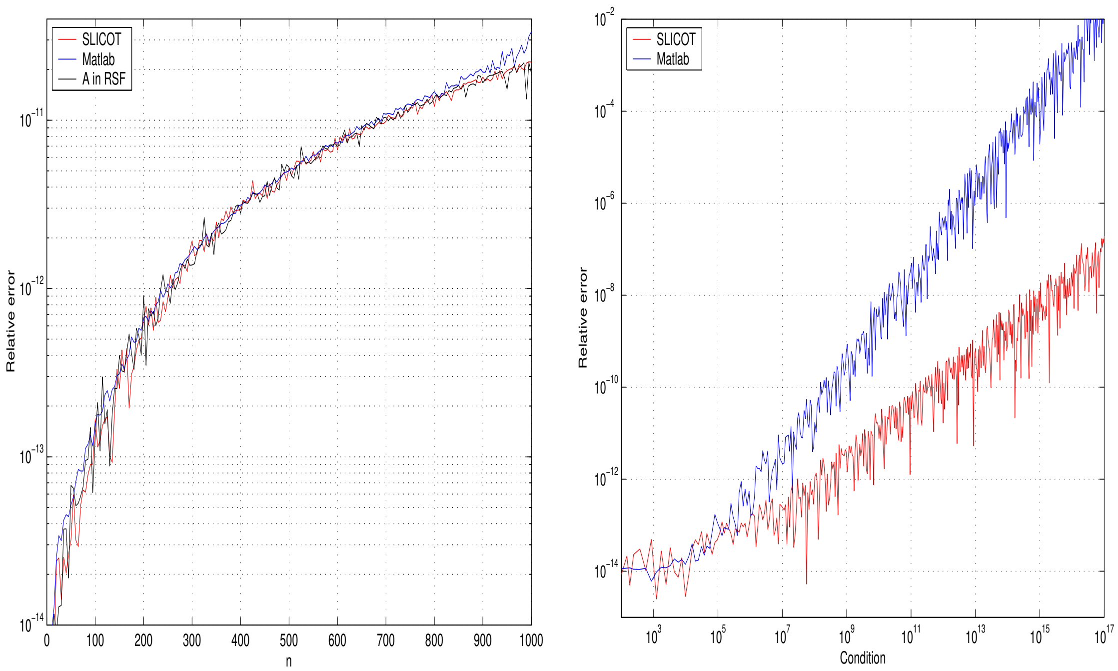


Figure 8: SLICOT **slstei** versus MATLAB **dlyap** for generated Stein equations using Example 4. The left plot shows the relative error comparison versus the dimension using the parameters $\lambda = -0.01$ and $s = 1.005$ while the right plot reveals the relative error versus the condition. For the latter, the following parameters are chosen: $\lambda = -0.9 \ldots 0$, $s = 1.2$ and $n = 10$.

SLICOT gains five digits in accuracy, see Figure 8 (right plot). Due to the fact that the accuracy of the SLICOT and MATLAB solver is nearly the same for Exemple 3 and 4, if A is in RSF, the behaviour of `slstei` is only presented in both plots.

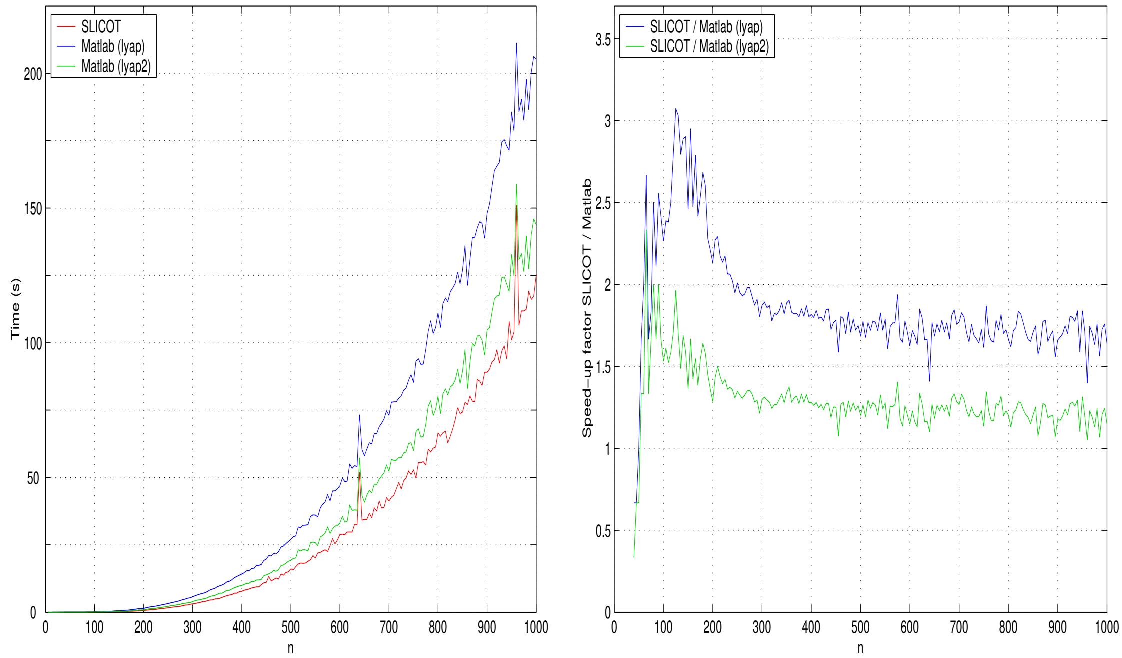


Figure 9: SLICOT `slstly` versus MATLAB `lyap` for generated stable Lyapunov equations using Example 5 with the parameters $a = 1.0129$, $s = 1.001$. Timing comparison (left) and speed-up factor (right).

Figure 9 shows the execution times for SLICOT function `slstly` and MATLAB functions `lyap` and `lyap2` (left plot), and the speed-up factor of `slstly` versus `lyap` and `lyap2` (right plot), for solving stable Lyapunov equations generated with Example 5 using the following parameters:

$$a = 1.0129, \quad s = 1.001, \quad n = 5 : 5 : 1000.$$

In this example, the SLICOT solver is between 1.7 and 3 times as fast as the MATLAB solver `lyap` and it is still between 1.2 and 2.2 times as fast as the MATLAB solver `lyap2`.

The relative error versus the dimension for the same example is shown in Figure 10 (left plot). Here, the MATLAB functions `lyap` and `lyap2` are combined with the MATLAB function `chol`, in order to show what can be gained from calculating the Cholesky factor directly. Although the accuracy of `lyap2` is better for $n < 500$, the SLICOT solver `slstly` reveals a better accuracy for $n > 500$. But note that the difference is marginal. Using Example 5 with these above-mentioned parameters, the condition is about 0.5. In order to vary the condition, the following parameters are used:

$$a = 5, \quad s = 1.02 \ldots 4.4, \quad n = 10.$$

In Figure 10 (right plot) the accuracy versus the condition is shown. It has to be mentioned that it is not possible to calculate the Cholesky factor from the solution of `lyap2` when the condition is higher than 5.0×10^{6}, because then the solution starts to contain negative eigenvalues. Therefore the graph ends at this point. The accuracy of the SLICOT solver `slstei` versus the MATLAB solvers `lyap` and `lyap2` is comparable and even sometimes better.

Figure 11 shows the execution times for the SLICOT function `slstst` and the MATLAB function `dlyap` (left plot), and the speed-up factor of `slstst` versus `lyap` (right plot), for solving stable Stein

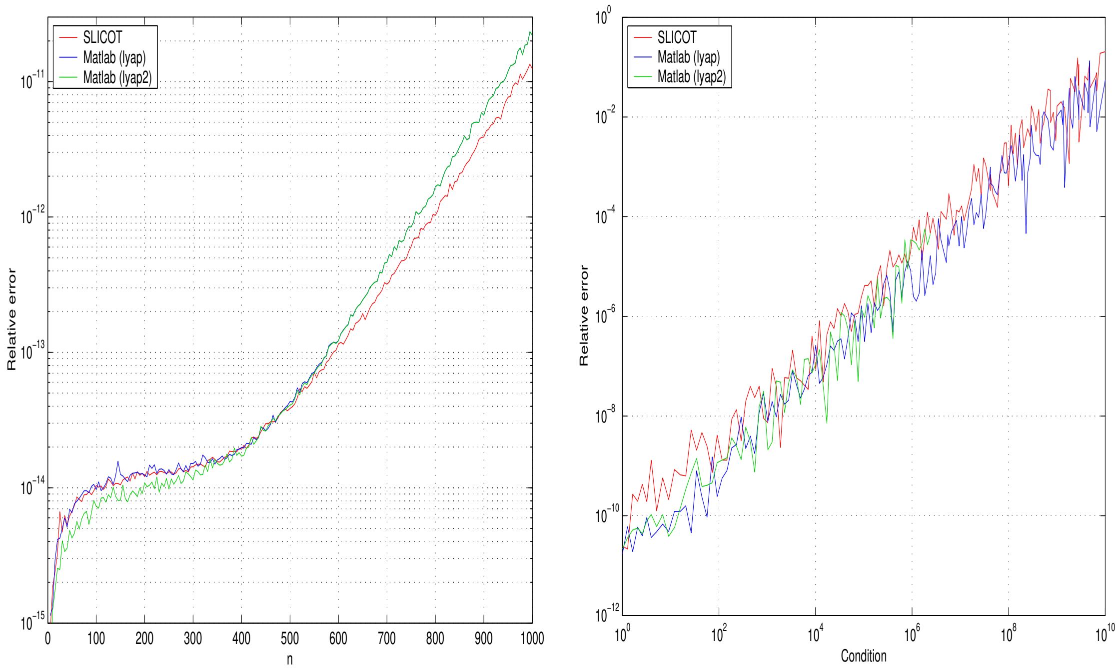


Figure 10: SLICOT **slstly** versus MATLAB **lyap** for generated stable Lyapunov equations with calculated Cholesky factor using Example 5. The left plot shows the relative error comparison versus the dimension using the parameters $a = 1.0129$ and $s = 1.001$ while the right plot reveals the relative error versus the condition. To achive this, we choose the following parameters: $a = 5$, $s = 1.02 \ldots 4.4$ and $n = 10$.

equations generated with Example 6 using the following parameters:

$$a = 1.001, \quad s = 1.01, \quad n = 5 : 5 : 1000.$$

These plots reveal that the SLICOT function `slstst` is between 1.8 and 3.2 times faster than the corresponding MATLAB function `dlyap`.

The relative error versus the dimension for the same example is shown in Figure 12 (left plot), whereat the condition varies between 1 and 1.5×10^6. The MATLAB function `dlyap` is combined with the MATLAB function `chol`, in order to show what can be gained from calculating the Cholesky factor explicitly. In order to vary the condition, the following parameters are used:

$$a = 1.5, \quad s = 1.01 \ldots 5.5, \quad n = 10.$$

While the SLICOT solver outperforms the MATLAB solver for stable Lyapunov equations, the accuracy is comparable, see Figure 12 (left plot), or even better, see Figure 12 (right plot) where SLICOT gains four digits.

Figure 13 shows the execution times for SLICOT function `slsylv` using the Hessenberg-Schur and the Schur method versus the MATLAB functions `lyap` and `lyap2` (left plot), and the speed-up factor of `slsylv` versus `lyap` and `lyap2` (right plot) for solving continuous-time Sylvester equations generated with Example 7 using the parameters:

$$a = 1.03, \quad b = 1.008, \quad s = 1.001, \quad n = 5 : 5 : 1000.$$

A closer look reveals that the execution times for the SLICOT function `slsylv` using the Schur method and for the MATLAB function `lyap` are comparable while the comparison with MATLAB's `lyap2` reveals

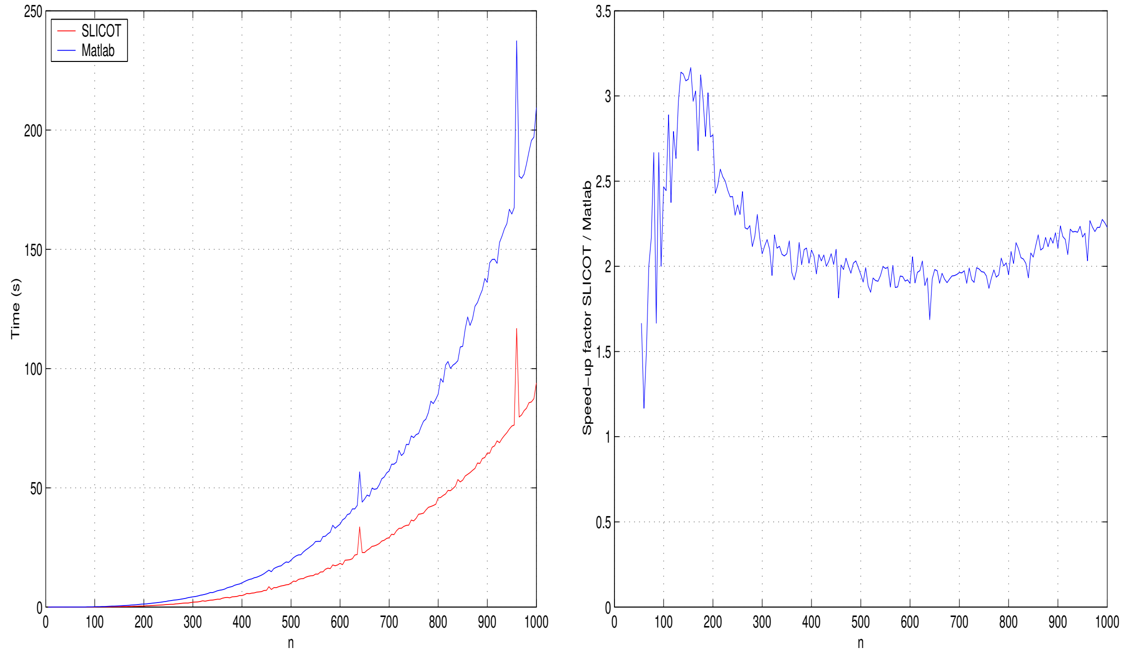


Figure 11: SLICOT **slstst** versus MATLAB **dlyap** for generated stable Stein equations using Example 6 with the parameters $a = 1.001$, $s = 1.01$. Timing comparison (left) and speed-up factor (right).

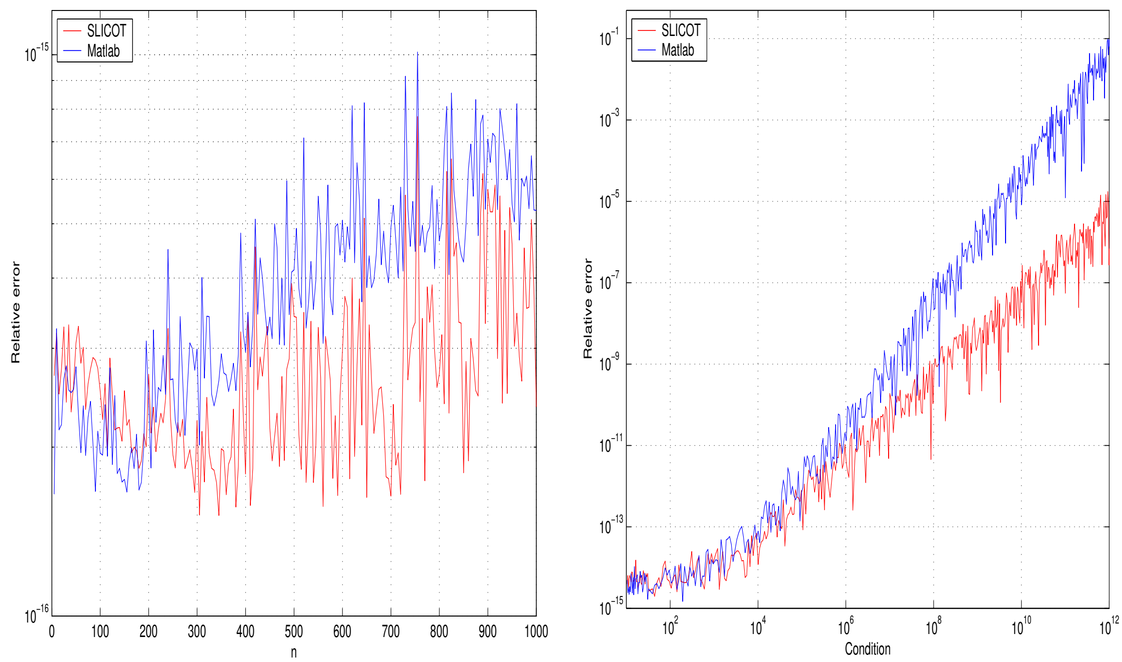


Figure 12: SLICOT **slstst** versus MATLAB **dlyap** for generated stable Stein equations with calculated Cholesky factor using Example 6. The left plot shows the relative error comparison versus the dimension using the parameters $a = 1.001$ and $s = 1.01$ while the right plot reveals the relative error versus the condition using the parameters $a = 1.5$, $s = 1.01 \ldots 5.5$ and $n = 10$.

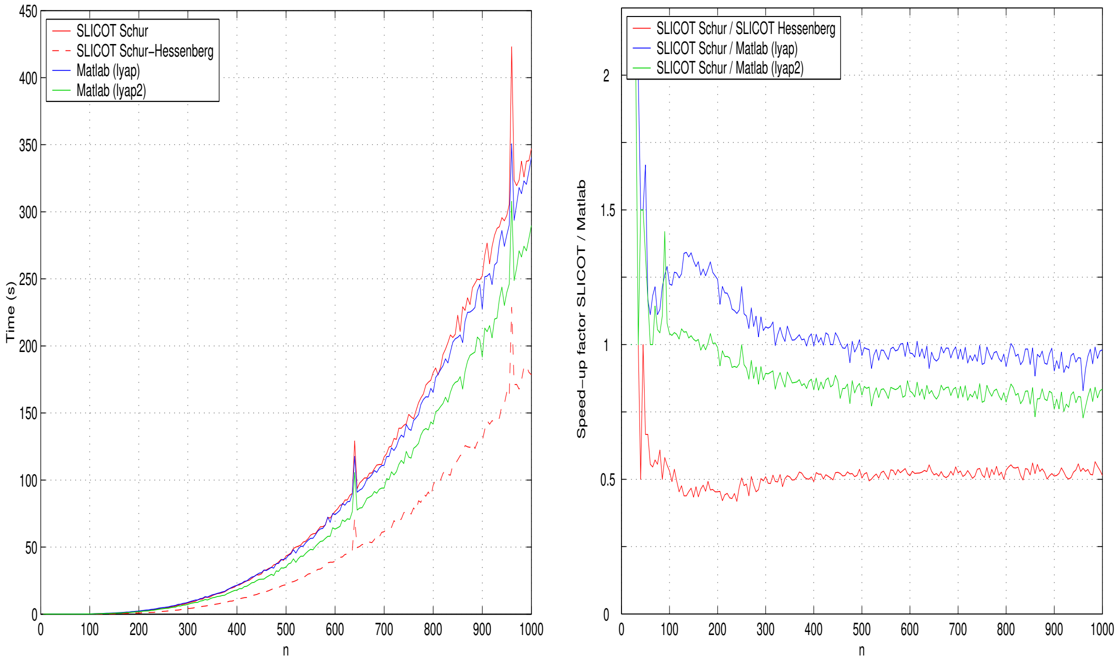


Figure 13: SLICOT `slsylv` Hessenberg-Schur method and Schur method versus MATLAB `lyap` and `lyap2` for generated continuous-time Sylvester equations using Example 7 with the parameters $a = 1.03$, $b = 1.008$, $s = 1.001$. Timing comparison (left) and speed-up factor (right).

that the SLICOT solver is about 20 % slower. Furthermore, the SLICOT function `slsylv` provides also the Hessenberg-Schur method to solve Sylvester equations, which is twice as fast as the Schur method, see Figure 13 (right plot). The relative error comparison versus the dimension, shown in Figure 14, is comparable for both methods. In order to obtain again a clear representation and due to the fact that the accuracy of both MATLAB solvers is similar, here we only present the results for `lyap`.

Up to Release 13, no MATLAB solver for discrete-time Sylvester equations was available. In order to show what is gained from a solver that directly treats the discrete-time equation rather than using a transformation to continuous-time, we choose an example with $(\pm 1 \notin \Lambda(A), \Lambda(B))$ so that it is possible to transform the discrete-time Sylvester equation into the continuous-time Sylvester equation:

$$\widetilde{A}X + X\widetilde{B} = -\widetilde{C}, \tag{40}$$

where

$$\widetilde{A} = (A+I)^{-1}(A-I), \quad \widetilde{B} = (B+I)(B-I)^{-1}, \quad \text{and} \quad \widetilde{C} = 2(A+I)^{-1}C(B-I)^{-1}.$$

So, Figure 15 shows the execution times for SLICOT function `sldsyl` using the Schur and the Hessenberg-Schur method and the MATLAB function `lyap` (left plot) solving the transformed equation, and the speed-up factor of `sldsyl` versus `lyap` (right plot) for solving discrete-time Sylvester equations generated with Example 8 using the parameters:

$$a = 1.005, \quad b = 1.01, \quad s = 1.005, \quad n = 5:5:1000.$$

It has to be mentioned that the Hessenberg-Schur method is up to three times faster than the corresponding MATLAB solver, while the Schur method implemented in the SLICOT solver is as fast as

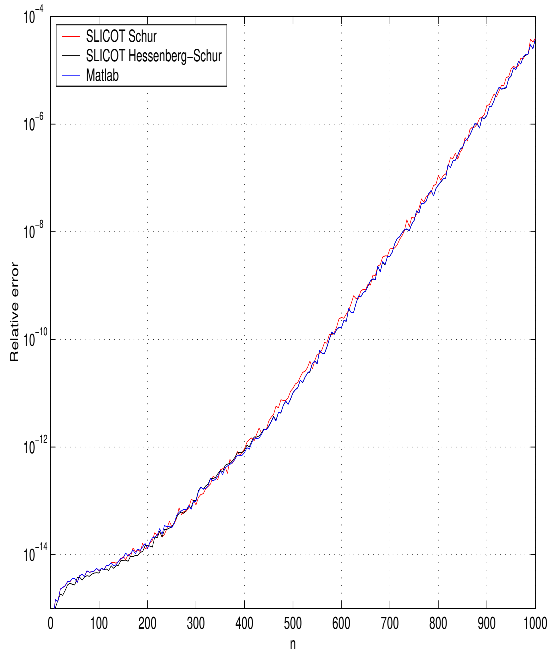


Figure 14: SLICOT `slsylv` Hessenberg-Schur method and Schur method versus MATLAB `lyap` for generated continuous-time Sylvester equations using Example 7. The plot shows the relative error versus the dimension using the parameters $a = 1.03$, $b = 1.008$ and $s = 1.001$.

the MATLAB solver. In Figure 16 the relative error versus the dimension is shown. While the accuracy of SLICOT's Schur and the Hessenberg-Schur method is comparable, the SLICOT solver `sldsyl` for discrete-time Sylvester equations gains up to three digits versus the MATLAB solver `lyap` for the transformed continuous-time Sylvester equation.

In the following paragraphs the results of the solvers for generalized Lyapunov and Sylvester equations will be discussed. Because of the fact that MATLAB (Release 13) does not provide any solver for this kind of equations, the examples are chosen in such a way that the generalized equations can be transformed into the standard Lyapunov and Sylvester equations. Hence, the generalized Lyapunov equations are transformed into:

$$\widetilde{A}^T X + X\widetilde{A} = -\widetilde{C} \tag{41}$$

$$\widetilde{A}^T X\widetilde{A} - X = -\widetilde{C}, \tag{42}$$

where

$$\widetilde{A} = A\,E^{-1} \quad \text{and} \quad \widetilde{C} = E^{-T}\,C\,E^{-1},$$

and the generalized stable Lyapunov equations are transformed into:

$$\widetilde{A}^T X + X\widetilde{A} = -\widetilde{C}^T\,\widetilde{C} \tag{43}$$

$$\widetilde{A}^T X\widetilde{A} - X = -\widetilde{C}^T\,\widetilde{C}, \tag{44}$$

where

$$\widetilde{A} = A\,E^{-1} \quad \text{and} \quad \widetilde{C} = C\,E^{-1}.$$

In order to solve the generalized Sylvester equation via MATLAB's `lyap`, we reduce the pair of equations by solving the second equation for Y and substitute it in the first equation. Then, we obtain the following

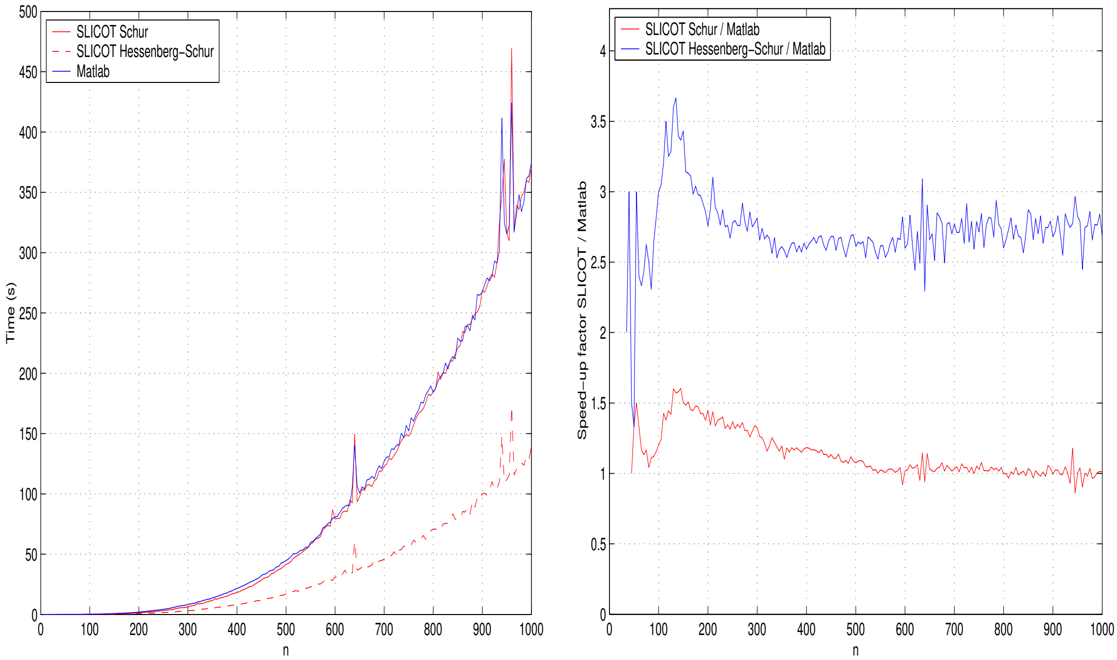


Figure 15: SLICOT **sldsyl** Hessenberg-Schur method and Schur method for generated discrete-time Sylvester equations versus MATLAB **lyap** for the transformed continuous-time Sylvester equations using Example 8 with the parameters $a = 1.005$, $b = 1.01$, $s = 1.005$. Timing comparison (left) and speed-up factor (right). Note that the execution time which MATLAB needs for solving the transformed Sylvester equation does not consider the time to obtain the transformed equations.

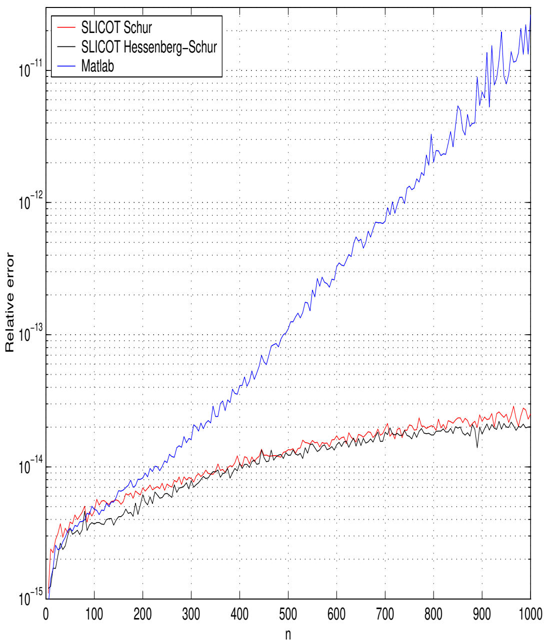


Figure 16: SLICOT **sldsyl** Hessenberg-Schur method and Schur method versus MATLAB **lyap** for generated discrete-time respectively transformed continuous-time Sylvester equations using Example 8. The plot shows the relative error versus the dimension using the parameters $a = 1.005$, $b = 1.01$ and $s = 1.005$.

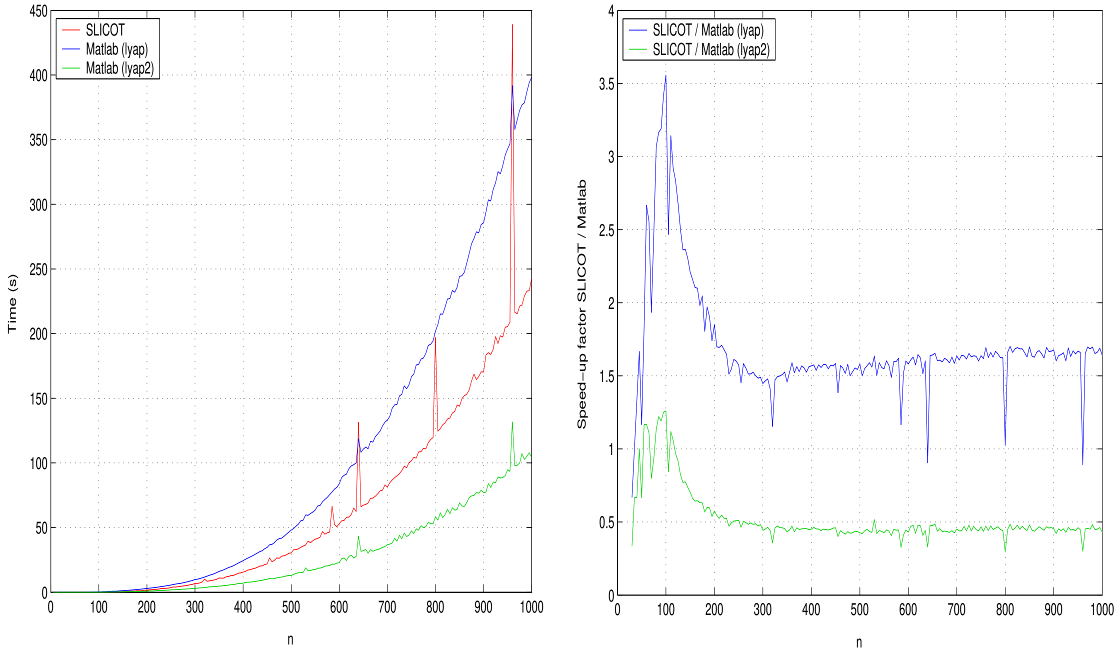


Figure 17: SLICOT **slgely** versus MATLAB **lyap** and **lyap2** for generated generalized respectively transformed Lyapunov equations using Example 9 with the parameter $t = 5$. Timing comparison (left) and speed-up factor (right). Note that the execution time which MATLAB needs for solving the generalized Lyapunov equation does not consider the time to obtain the transformed equations.

continuous-time Sylvester equation:

$$\widetilde{A}X + X\widetilde{B} = \widetilde{C}, \qquad Y = EXF^{-1} - HF^{-1}, \tag{45}$$

where

$$\widetilde{A} = -E^{-1}A, \quad \widetilde{B} = F^{-1}B, \quad \text{and} \quad \widetilde{C} = -E^{-1}G + E^{-1}HF^{-1}B.$$

Figure 17 shows the execution times for SLICOT function `slgely` and MATLAB functions `lyap` and `lyap2` (left plot), and the speed-up factor of `slgely` versus `lyap` and `lyap2` (right plot), for solving generalized Lyapunov equation generated with Example 9 using the following parameters:

$$t = 5, \quad n = 5 : 5 : 1000.$$

Note that for solving the transformed Lyapunov equation with MATLAB, the execution time does not consider the time to generate the needed equation. Hence, the execution time comparison is just a reference in order to judge the efficiency of the SLICOT solver. A closer look at the speed-up factor reveals that SLICOT solver `slgely` is between 1.5 and 3.3 times faster than MATLAB `lyap`, especially for dimensions till $n = 200$. It is interesting to notice that even though `slgely` solves a generalized Sylvester equation while `lyap` solves a standard Lyapunov equation, it is still faster. Furthermore SLICOT and MATLAB show again peaks at $n = 640$ and $n = 960$ caused by cache effects, see Figure 17 (left plot). But it has to be mentioned, that the peaks caused by the SLICOT solver are up to 5 times higher and in addition there are two peaks at $n = 590$ and $n = 800$. The difference is due to the fact that `slgely` needs two data matrices, A and E, while `lyap2` uses only one data matrix, $\widetilde{A}$. Furthermore it has to be mentioned that the MATLAB function `lyap2` is more than twice as fast as the SLICOT function `slgely`, if the dimension is higher than $n = 200$.

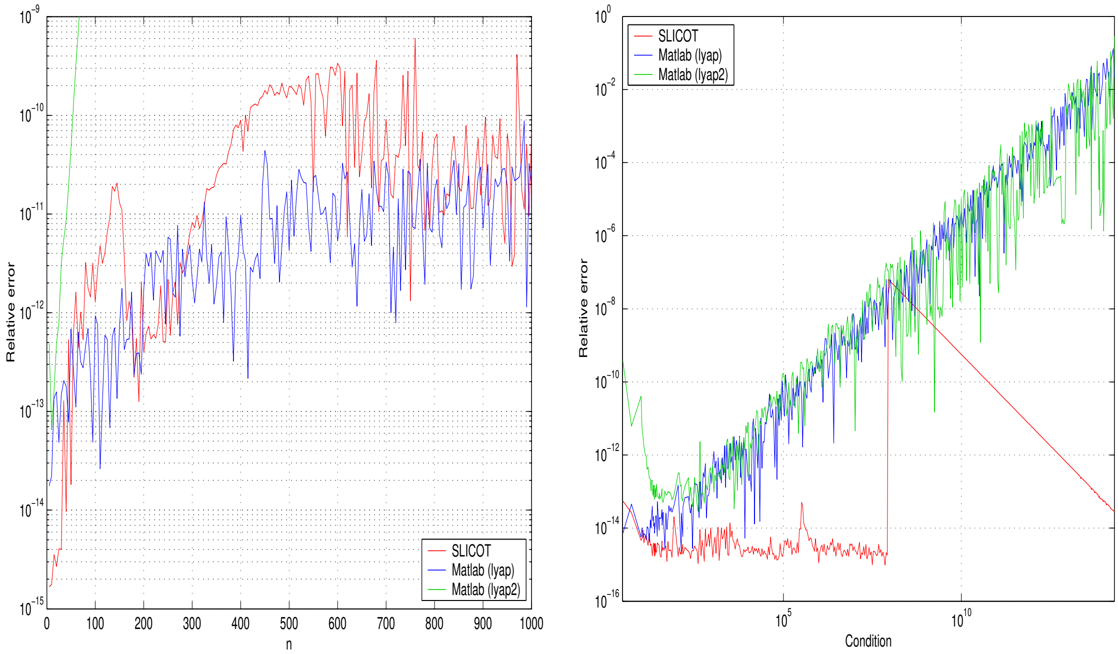


Figure 18: SLICOT **slgely** versus MATLAB **lyap** and **lyap2** for generated generalized respectively transformed Lyapunov equations using Example 9. The left plot shows the relative error comparison versus the dimension using the parameter $t = 5$ while in the right plot we see the relative error versus the condition. To achieve this, we use the following parameters: $t = 1.05 \ldots 46$ and $n = 10$.

In Figure 18 (left plot) the relative error versus the dimension for the same example is shown. Although SLICOT gains up to two digits for small dimension (till $n = 50$), tendencially SLICOT loses between one or two digits. Indeed `lyap2` outperforms `slgely`, but it looses rapidly up to three digits in accuracy till dimension $n = 70$. So in order to solve this example, `lyap2` is not useful. It should be remarked that the condition in this example varies between 60 and 170. Figure 18 (right plot) shows the relative error comparison versus the condition using Example 9 with the parameters:

$$t = 1.05 \ldots 46, \quad n = 10,$$

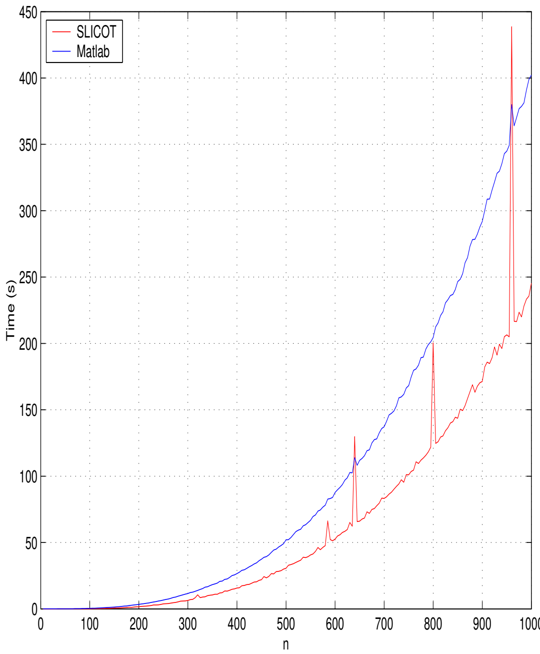
where the SLICOT solver for generalized Lyapunov equations gains up to nine digits in accuracy. It is curious that the relative error of the SLICOT solver varies little while the condition number is increased from 10^1 to 10^8, but then the relative error jumps over ten digits and decreases linearly afterwards.

The efficiency of SLICOT's generalized discrete-time Lyapunov solver is up to 3.8 times better than the MATLAB solver while the accuracy is comparable, which is shown in Figure 19 and Figure 20 (left plot). Figure 19 reveals the execution times for SLICOT function `slgest` and MATLAB function `dlyap` (left plot) solving the transformed equation, and the speed-up factor of `slgest` versus `dlyap` (right plot), for solving generalized Stein equations generated with Example 10 using the following parameters:

$$t = 5, \quad n = 5 : 5 : 1000.$$

Figure 20 (left plot) shows the relative error versus the dimension for the same example. Using these parameters, the condition varies between 30 and 65. In the right plot of this figure an upper bound for the relative error versus the condition is presented using Example 10 with the parameters:

$$t = 1.1 \ldots 36, \quad n = 10.$$



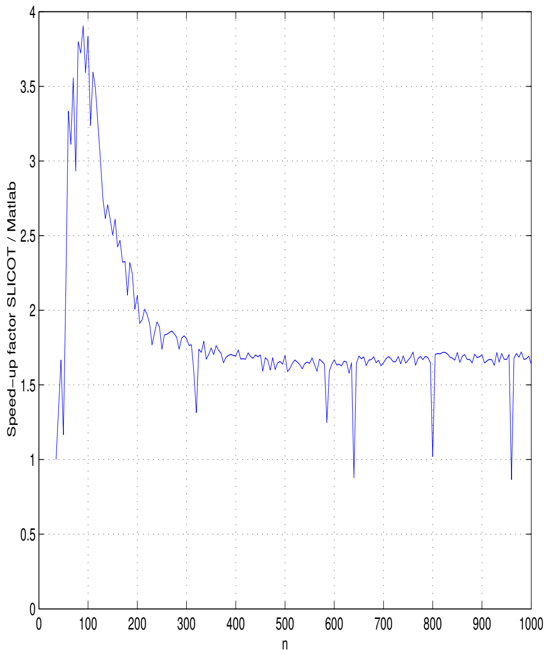
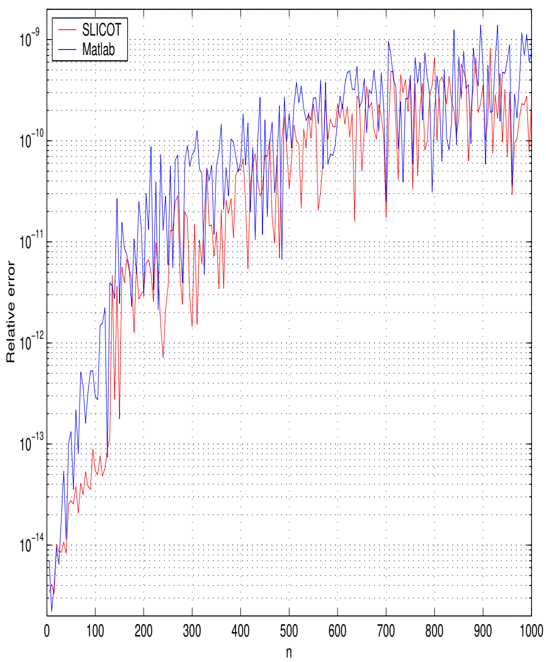


Figure 19: SLICOT **slgest** versus MATLAB **dlyap** for generated generalized respectively transformed Stein equations using Example 10 with the parameter $t = 5$. Timing comparison (left) and speed-up factor (right). Note that the execution time which MATLAB needs for solving the generalized Stein equation does not take into account the time to obtain the transformed equations.



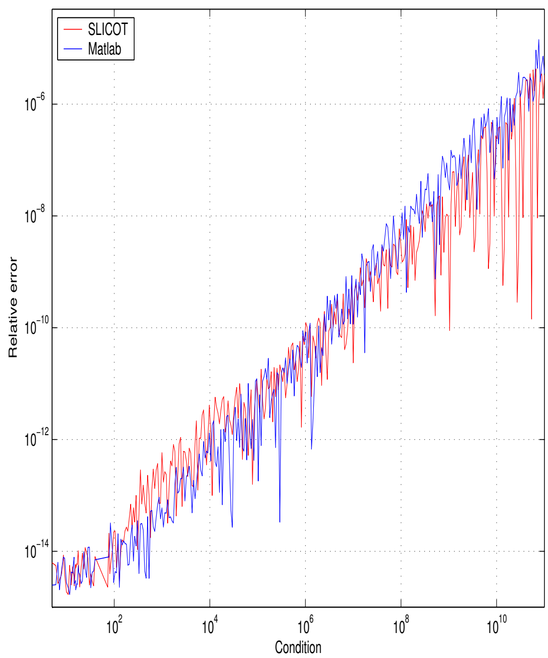


Figure 20: SLICOT **slgest** versus MATLAB **dlyap** for generated generalized respectively transformed Stein equations using Example 10. The left plot shows the relative error comparison versus the dimension using the parameter $t = 5$ while the right plot reveals the relative error versus the condition. For the latter, the following parameters are chosen: $t = 1.1 \ldots 36$ and $n = 10$.

The accuracy of the SLICOT and MATLAB solvers is similar.

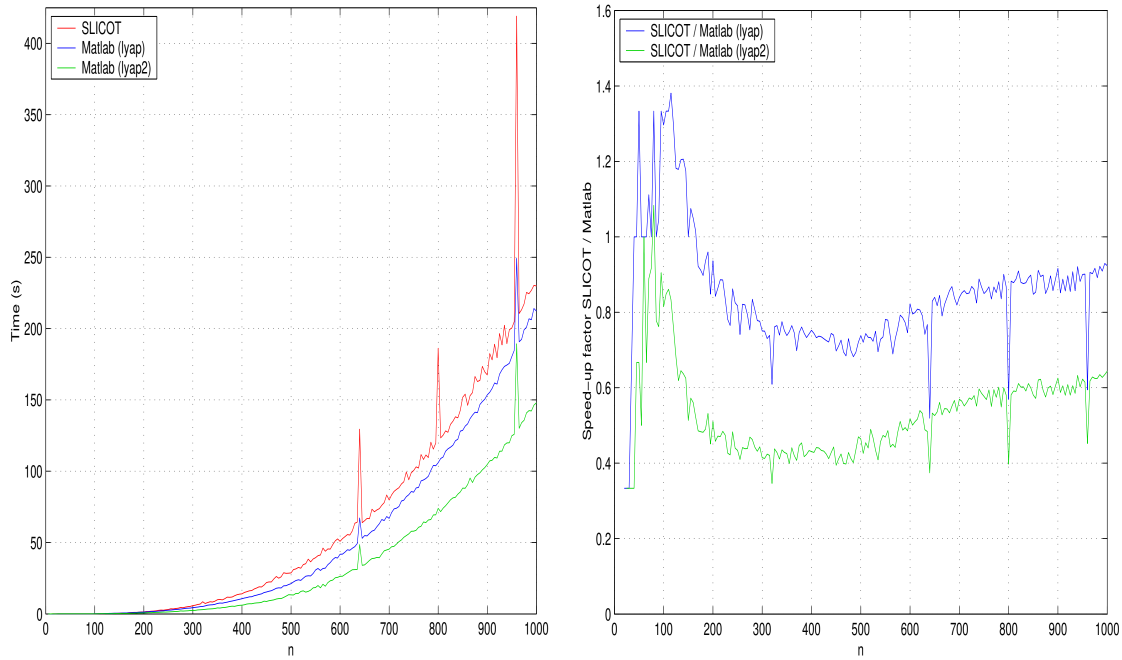


Figure 21: SLICOT `slgsly` versus MATLAB `lyap` and `lyap2` for generated generalized respectively transformed stable Lyapunov equations using Example 11 with the parameters $a = 1.005$, $e = 1.003$ and $s = 1.01$. Timing comparison (left) and speed-up factor (right). Note that the execution time which MATLAB needs for solving the generalized stable Stein equation does not take into account the time to obtain the transformed equation and the Cholesky factor of the solution.

Figure 21 shows the execution times for SLICOT function `slgsly` and MATLAB functions `lyap` and `lyap2` (left plot) solving the transformed equation, and the speed-up factor of `slgsly` versus `lyap` and `lyap2` (right plot), for solving generalized stable Lyapunov equation generated with Example 11 using the following parameters:

$$a = 1.005, \quad e = 1.003, \quad s = 1.01, \quad n = 5 : 5 : 1000.$$

Note that the execution times neither take into account the time to generate the transformed equations for MATLAB nor the time to calculate the Cholesky factor of the solution. Hence, this execution time comparison is just a reference in order to judge the efficiency of the SLICOT solver. It is amazing that the SLICOT solver `slgsly` for generalized stable Lyapunov equations is only about 20 % slower than the MATLAB solver `lyap` for standard Lyapunov equations although it is based on the QZ algorithm which is 2–3 times more expensive than the QR algorithm used by `lyap`. In comparison to MATLAB solver `lyap2`, SLICOT is about 50 % slower. Figure 22 (left plot) shows that the accuracy of these solvers is comparable whereas up to dimension $n = 650$ `slgsly` is better than MATLAB solvers but it is a little worser for higher dimension. Note that the condition varies between 0.2 and 1.45×10^6 in this example with the above-mentioned parameters. But the Figure 22 (right plot), which shows the relative error comparison versus the condition using Example 11 with the parameters:

$$a = 1.1, \quad e = 1.01, \quad s = 1.51 \ldots 7, \quad n = 10\,,$$

shows that SLICOT gains up to four digits in accuracy.

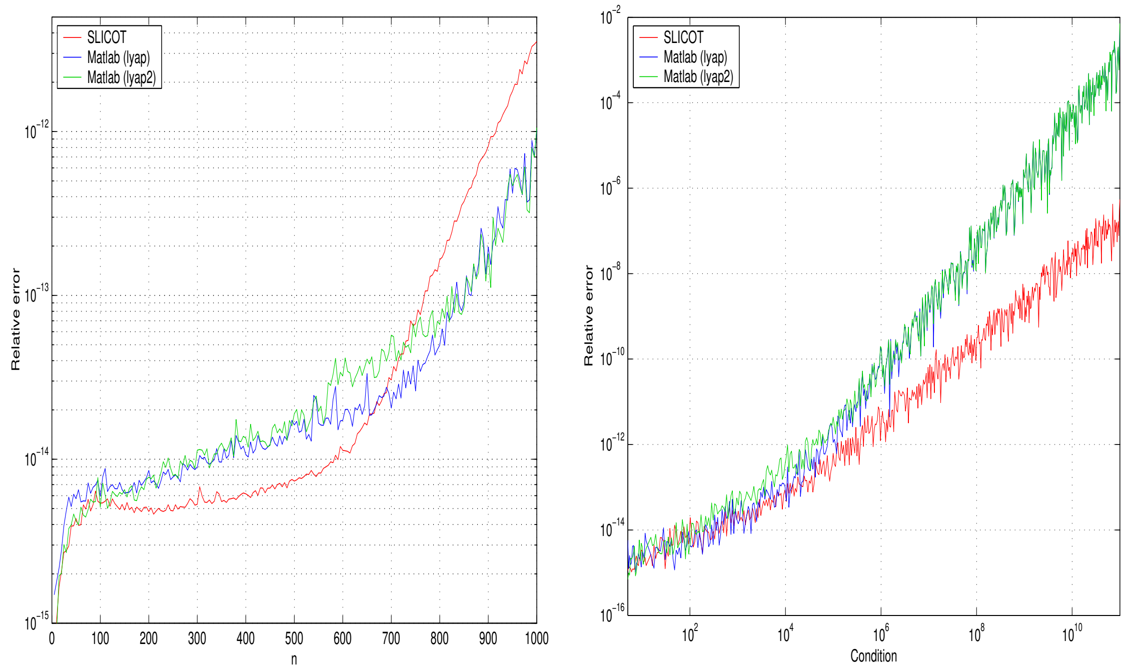


Figure 22: SLICOT `slgsly` versus MATLAB `lyap` and `lyap2` for generated generalized respectively transformed stable Lyapunov equations using Example 11. The left plot shows the relative error versus the dimension using the parameters $a = 1.005$, $e = 1.003$ and $s = 1.01$ while the right plot presents the relative error versus the condition. In order to achieve this, we choose the following parameters: $a = 1.1$, $e = 1.01$, $s = 1.51 \ldots 7$ and $n = 10$.

Analogous results are also obtained for generalized stable Stein equations using Example 12 with the parameters:

$$a = 1.005, \quad e = 1.003, \quad s = 1.01, \quad n = 5 : 5 : 1000.$$

Figure 23 shows the execution times for SLICOT function `slgsst` and MATLAB function `dlyap` (left plot) solving the transformed equation, and the speed-up factor of `slgsst` versus `dlyap` (right plot). Again it has to be mentioned that the execution time, needed by MATLAB for the transformed problem, neither takes into account the time to generate this transformed equations nor the time to calculate the Cholesky factor of the solution. Hence, the SLICOT solver `slgsst` is slower than MATLAB, while the accuracy is comparable, whereat the condition varies between 0.5 and 3.0×10^7. This is shown in Figure 24 (left plot), where the relative error comparison versus the dimension is represented. But the Figure 24 (right plot), which shows the relative error comparison versus the condition using Example 12 with the parameters:

$$a = 1.1, \quad e = 1.01, \quad s = 1.51 \ldots 8, \quad n = 10\,,$$

reveals that SLICOT gains up to seven digits in accuracy,

In the following the results for the generalized Sylvester equation solvers are presented. For this purpose, the Example 13 with the parameters

$$a = 1.001, \quad b = 1.002, \quad e = 1.003, \quad f = 1.004, \quad s = 1.01, \quad n = 5 : 5 : 1000$$

is employed. Figure 25 represents the execution times for SLICOT function `slgesg` and MATLAB function `lyap` and `lyap2` (left plot), and the speed-up factor of `slgesg` versus `lyap` and `lyap2` (right plot),

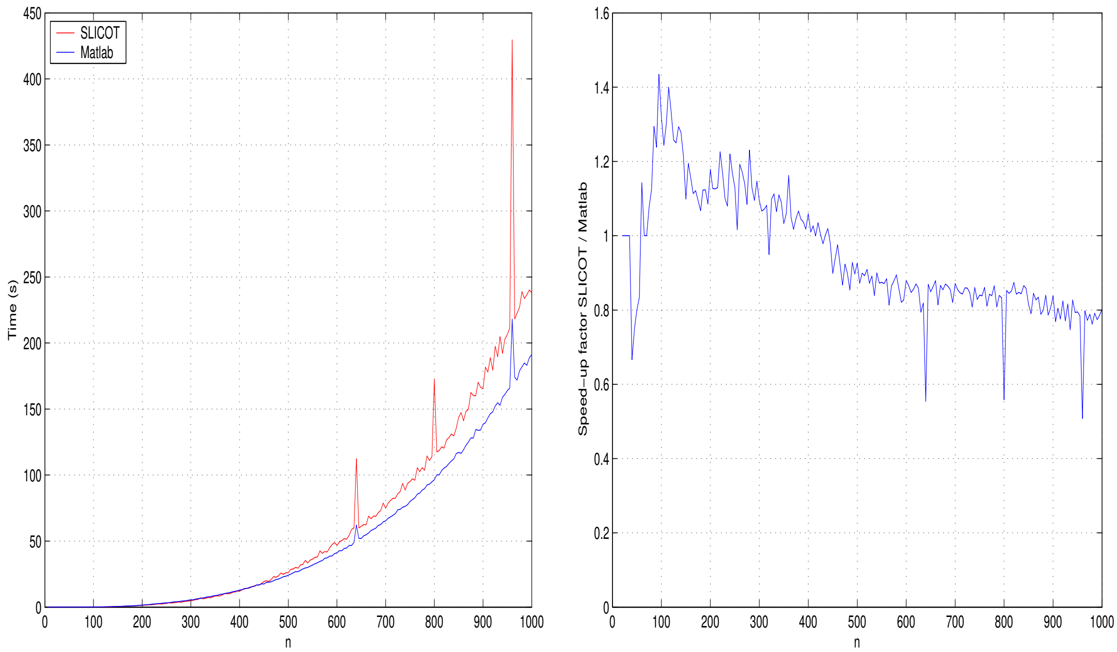


Figure 23: SLICOT **slgsst** versus MATLAB **dlyap** for generated generalized respectively transformed stable Stein equations using Example 12 with the parameters $a = 1.005$, $e = 1.003$ and $s = 1.01$. Timing comparison (left) and speed-up factor (right). Note that the execution time does not consider the time to obtain the transformed equation for MATLAB or to calculate the Cholesky factor of the solution.

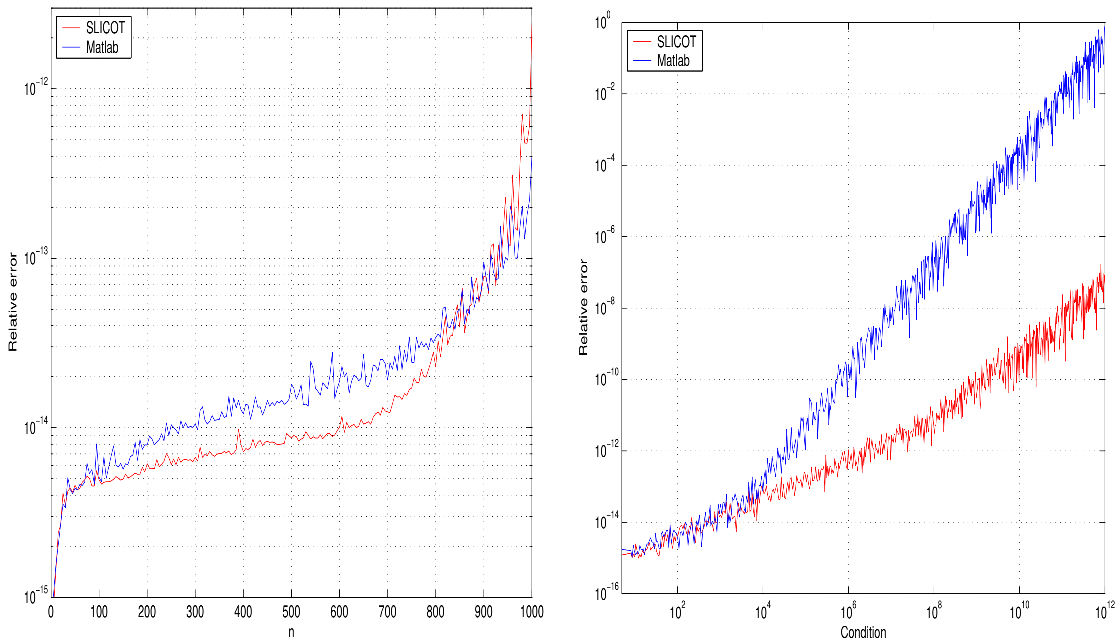


Figure 24: SLICOT **slgsst** versus MATLAB **dlyap** for generated generalized respectively transformed stable Stein equations with calculated Cholesky factor using Example 12. The left plot shows the relative error versus the dimension using the parameters $a = 1.005$, $e = 1.003$ and $s = 1.01$ and the right plot reveals the relative error versus the condition using $a = 1.1$, $e = 1.01$, $s = 1.51 \ldots 8$ and $n = 10$.

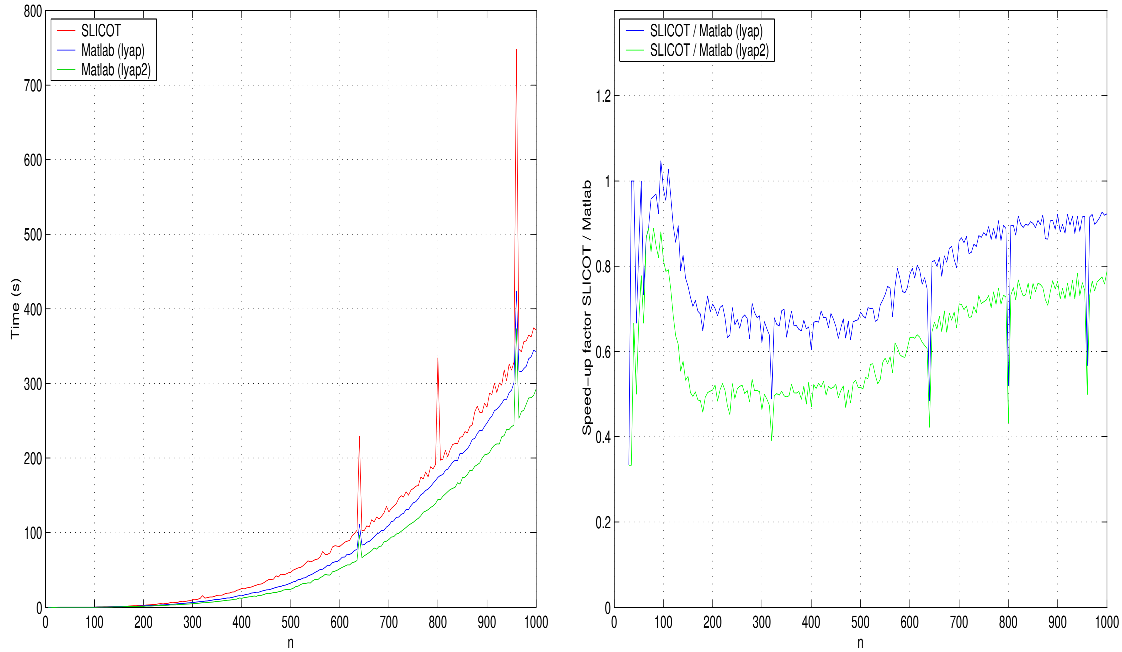


Figure 25: SLICOT **slgesg** versus MATLAB **lyap** and **lyap2** for generated generalized Sylvester equations using Example 13 with the parameters $a = 1.001$, $b = 1.002$, $e = 1.003$, $f = 1.004$, $s = 1.01$. Timing comparison (left) and speed-up factor (right). Note that the execution time which MATLAB needs for solving the generalized Sylvester equation does not take into account the time to obtain the transformed equation.

for solving the generalized Sylvester equation respectively the transformed standard continuous-time Sylvester equation. Again, it is amazing that the SLICOT solver for the generalized Sylvester equations is only a little slower than the MATLAB solver for the standard Sylvester equation, if the dimension is higher than $n = 700$. Between the dimensions $n = 150$ and $n = 500$, `slgesg` is about 30 % slower than `lyap` and about 50 % slower than `lyap2`. Apart from the three peaks at $n = 640$, $n = 800$ and $n = 960$ the SLICOT function `slgesg` is almost as fast as the MATLAB function for the corresponding transformed equation, especially for $n = 1000$, where the difference amounts 10 % (`lyap`) or 20 % (`lyap2`), respectively.

Figure 26 (left plot) shows the relative error comparison versus the dimension for the same example. A closer look reveals that until $n = 800$ the difference in the accuracy for the $X-$ and $Y-$solution produced by SLICOT and MATLAB is marginal. But, starting at $n = 800$, SLICOT loses up to two digits. Note that the condition varies between 0.6 and 53. In the right plot of Figure 26 the relative error versus the condition is presented using Example 13 with the parameters:

$$\begin{aligned} a = 1.001, \quad b &= 1.002, \quad e = 1.003, \quad f = 1.004, \\ s &= 1.03 \ldots 11, \quad n = 10. \end{aligned}$$

There SLICOT gains up to four digits for the $X-$solution and up to five digits for the $Y-$solution in accuracy.

6 Conclusions

We have discussed easy-to-use solvers from the SLICOT Library for various linear matrix equations from systems and control theory. Based on Fortran 77 codes implementing state-of-the-art numerical algo-

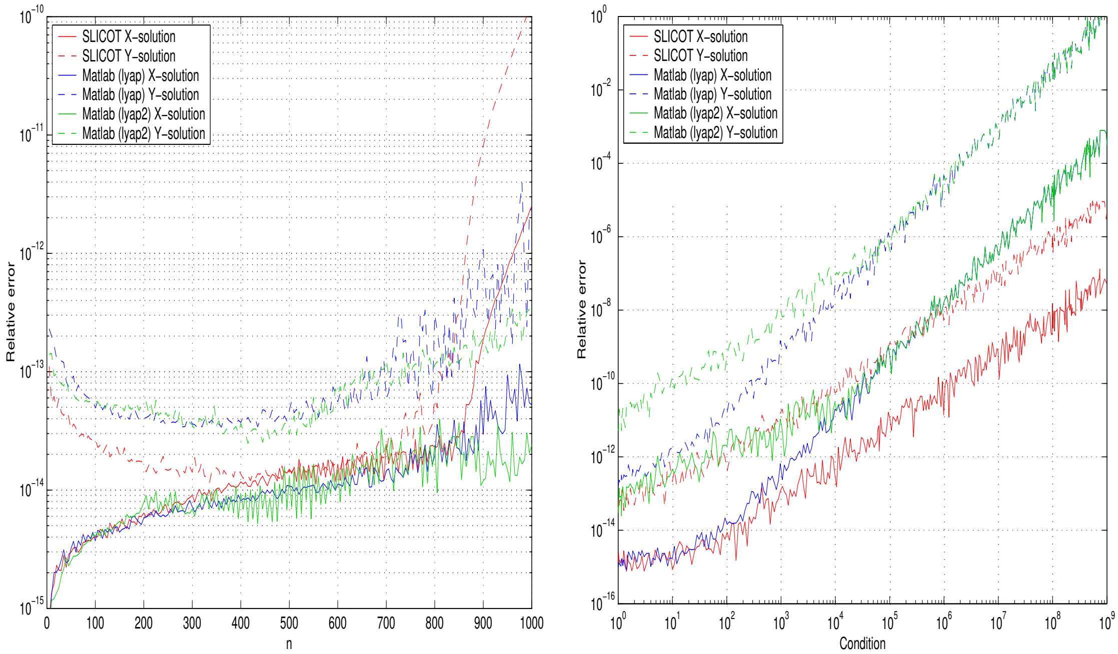


Figure 26: SLICOT **slgesg** versus MATLAB **lyap** and **lyap2** for generated generalized or transformed Sylvester equations using Example 13. The left plot shows the relative error comparison versus the dimension for $a = 1.001$, $b = 1.002$, $e = 1.003$, $f = 1.004$, $s = 1.01$ while the right plot reveals the relative error versus the condition for $a = 1.001$, $b = 1.002$, $e = 1.003$, $f = 1.004$, $s = 1.03 \ldots 11$ and $n = 10$.

rithms, the high-level MATLAB or Scilab interfaces offer extended functionality, and improved efficiency and comparable reliability over existing software tools, as illustrated by the numerical results. Even though parts of the SLICOT solvers have been integrated into MATLAB's Release 14 version of the Control Toolbox, there are still advantages in using the original SLICOT tools as a larger class of matrix equations can be solved (generalized Sylvester equations) and the sensitivity of numerical computations can be accessed via condition estimators.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide: Third Edition.* Software · Environments · Tools. SIAM, Philadelphia, PA, 1999.

[2] A.C. Antoulas. *Lectures on the Approximation of Large-Scale Dynamical Systems.* SIAM, Philadelphia, PA, to appear.

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

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

[5] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT — A subroutine library in systems and control theory. In B. N. Datta, Ed., *Applied and Computational Control, Signals, and Circuits*, volume 1, chapter 10, pages 499–539. Birkhäuser, Boston, 1999.

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

[7] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Lin. Alg. Appl.*, 85(1):267–279, 1987.

[8] C. Gomez (Ed.). *Integrated Scientific Computing with Scilab*. Birkhäuser, Boston, 1997.

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

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

[11] W. Enright, *Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations*, ACM Trans. Math. Softw., 4 (1978), pp. 127–136.

[12] M. Epton, *Methods for the solution of $AXD - BXC = E$ and its application in the numerical solution of implicit ordinary differential equations*, BIT, 20 (1980), pp. 341–345.

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

[14] S. J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[15] B. Kågström and P. Poromaa. 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, 1996.

[16] D. Kreßner, V. Mehrmann, and T. Penzl CTLEX – a Collection of Benchmark Examples for Continuous-Time Lyapunov Equations. SLICOT Working Note 1999-6

[17] D. Kreßner, V. Mehrmann, and T. Penzl DTLEX – a Collection of Benchmark Examples for Discrete-Time Lyapunov Equations. SLICOT Working Note 1999-7

[18] A. J. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Trans. Automat. Control*, AC–24(6):913–921, 1979.

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

[20] The MathWorks, Inc, 24 Prime Park Way, Natick, Mass., 01760–1500. *Control System Toolbox User's Guide*, 1998. For use with MATLAB.

[21] The MathWorks, Inc, 24 Prime Park Way, Natick, Mass., 01760–1500. *Using MATLAB. Version 5*, 1999.

[22] G. Obinata and B.D.O. Anderson. *Model Reduction for Control System Design.* Communications and Control Engineering Series, Springer-Verlag, London, UK, 2001.

[23] T. Pappas, A. J. Laub, and N. R. Sandell. On the numerical solution of the discrete-time algebraic Riccati equation. *IEEE Trans. Automat. Control*, AC–25(4):631–641, 1980.

[24] T. Penzl. Numerical solution of generalized Lyapunov equations. *Advances in Comp. Math.*, 8:33–48, 1998.

[25] V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics: A Series of Monographs and Textbooks*, E.J. Taft and Z. Nashed (Series Eds.). Marcel Dekker, Inc., New York, 1996.

[26] V. Sima, P. Petkov, and S. Van Huffel. 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, Algebraic Systems Theory (3), 10 pages.

[27] P. Van Dooren. A generalized eigenvalue approach for solving Riccati equations. *SIAM J. Sci. Stat. Comput.*, 2(2):121–135, 1981.

[28] S. Van Huffel and V. Sima. SLICOT and control systems numerical software packages. In P.R. Kalata, Ed., *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., pages 39–44. Omnipress, 2002.