SLICOT Linear Systems Identification Toolbox *

Vasile Sima †‡

July 2000

*This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001-Leuven-Heverlee, BELGIUM. This report is also available by anonymous ftp from `wgs.esat.kuleuven.ac.be` in the directory `pub/WGS/REPORTS/SLWN2000-4.ps.Z`

†National Institute for Research & Development in Informatics, 71316 Bucharest 1, Romania and
‡ESAT - Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001-Leuven-Heverlee, Belgium

Contents

Abstract

Basic algorithmic, implementation and numerical issues involved in the new, subspace-based linear multivariable system identification toolbox—SLIDENT—incorporated in the freely available Subroutine Library in Control Theory (SLICOT) are described. Reliability, efficiency, and ability to solve large, industrial identification problems received a special consideration in the toolbox development process. Flexibility of usage is enhanced by providing various options: two algorithmic subspace-based approaches (MOESP and N4SID) and their combination, standard or fast techniques for data compression (including abilities to exploit the block-Hankel structure), multiple (possibly connected) data batches processing, availability of both, fully documented drivers and computational routines, the use of structure exploiting algorithms and dedicated linear algebra tools, etc. In addition, optionally, the quality of the intermediate results can be assessed by inspecting the associated condition numbers. The mexfiles developed based on the SLICOT Fortran subroutines enable to combine the efficiency of Fortran calculations with the simplicity of use in a MATLAB-like environment. Related m-files are also included in the toolbox, in order to increase the user-friendliness. Extensive comparisons with the available subspace techniques have been made. The numerical results show that SLICOT system identification toolbox—SLIDENT—is highly performant.

This report summarizes the achievements and deliverables of Task III.A of the NICONET Project. After a short description of the linear system identification problem and of the available subspace-based techniques to solve it, the numerical algorithms implemented in SLIDENT are surveyed. The associated Fortran routines are then listed and their functional abilities are outlined. The developed interfaces to MATLAB or Scilab, as well as examples of use are presented. Comparisons with the available MATLAB codes are included, illustrating the efficiency and accuracy of the SLIDENT components.

1 Introduction

Consider a linear time invariant (LTI) state space model, described by

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k, \\ y_k &= Cx_k + Du_k + v_k, \end{aligned} \tag{1}$$

where x_k is the n-dimensional state vector at time k, u_k is the m-dimensional input (control) vector, y_k is the ℓ-dimensional output vector, $\{w_k\}$ and $\{v_k\}$ are state and output disturbance or noise sequences, and A, B, C, and D are real matrices of appropriate dimensions. In *system identification* problems, the system order, n, and the quadruple of system matrices (A, B, C, D) have to be determined; the only available information is given by an upper bound, s, on n, and by the input and output data sequences, $\{u_k\}$ and $\{y_k\}$, $k = 1\colon t$ (i.e., for k taking integer values from 1 to a given t). In addition, the Kalman gain matrix, as well as the state and output covariance matrices have often to be found.

Two basic subspace-based approaches have been proposed for solving system identification problems: MOESP and N4SID. The main feature of the MOESP class of techniques is the determination of an extended observability matrix of the deterministic part of the model (1). The basic idea was first described in [22]. The main feature of the N4SID class of techniques is the determination of the state sequence of the LTI system to be identified, or of an observer to reconstruct its state sequence, via the intersection of the row spaces of the Hankel-like matrices constructed from "past" and "future" I/O data. The basic idea was described in [12], and extended in [20, 21]. See also the recent survey [3].

Both MOESP and N4SID associated algorithms start by building a *block-Hankel-block matrix* H, from two concatenated block-Hankel matrices, using the available input-output data, and finding an upper triangular factor from a QR (or RQ) factorization of H. MATLAB[1] and Fortran codes implementing these approaches have been developed, e.g., [19, 23, 8, 18]. The Fortran codes described in [18, 13, 14] include an option to process the input-output data either in a single, or in multiple data "batches", which is important from a practical point of view. These two strategies are referred to as *non-sequential*, and *sequential data processing*, respectively.

New codes, implementing the subspace methods for linear system identification, have been written for the SLICOT Library [2], in the framework of the NICONET project. SLICOT is based on the linear algebra package LAPACK [1], and the BLAS collections [10, 6, 5]. The new Fortran routines include algorithmic improvements and refinements over the previous implementations. For instance, it is possible to exploit the particular structure of the block-Hankel-block matrix H, when computing the R factor of its QR factorization, and this could speed-up the calculations by more than an order of magnitude. Special problem structure is exploited whenever feasible. In addition, more flexibility is offered. The SLICOT collection allows to combine the system identification techniques, as proposed in [7]. For instance, it is possible to estimate the matrices A and C using the MOESP approach, and the matrices B and D using either the N4SID approach, or an alternate technique, which resorts again to the input and output records (Subsection 2.4).

In order to increase the user-friendliness, MATLAB interfaces, represented by either mexfiles

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

or m-files, are provided. These interfaces have been designed to allow almost the same flexibility as the underlying Fortran routines, but many parameters are represented as options with default values. The MATLAB interfaces have been also used to build similar Scilab [4] interfaces to the SLICOT identification routines.

Basic algorithmic, implementation and numerical issues involved in the SLICOT subspace-based linear multivariable system identification toolbox—SLIDENT—are described in this report. Two numerical techniques for data compression are investigated and compared: the standard QR factorization of the matrix H, and the Cholesky factorization of the associated (inter-)correlation matrix, built exploiting the block-Hankel structure. In addition, a fast QR factorization technique [9], based on the displacement structure, is incorporated in the toolbox. Then, the associated Fortran routines are described and their functional abilities are outlined. The report also summarizes the functions and the structure of the MATLAB/Scilab interface of the toolbox, and presents some examples of use and numerical results obtained using the toolbox components. The comparisons with the available MATLAB identification tools show that SLICOT identification toolbox delivers equally accurate results, but often with much higher efficiency than other toolboxes.

2 SLICOT Identification Toolbox: Algorithmic Outline

2.1 Input-Output Data Processing using Subspace Techniques

The data used by the SLICOT toolbox for subspace identification is basically given as two $t \times m$ and $t \times \ell$ matrices, U and Y, for inputs and outputs, respectively, where t is the number of available measurements. So, the rows in U and Y are all the inputs or outputs measured at a certain time. Specifically,

$$U = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_t^T \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_t^T \end{bmatrix}.$$

In some applications, the number t is too large, and all data samples cannot be processed as a single batch. For instance, the available memory space could not be sufficient to accommodate for all these data, or the data themselves come in batches. Therefore, provisions have been taken to enable sequential processing of the input/output measurements. In this case, each new data batch is processed by updating a compressed representation of all the data in the previous batches.

For standard non-sequential data processing, the $N \times (m+\ell)p$ block-Hankel-block matrix $H = \begin{bmatrix} U_{1,p,N}^T & Y_{1,p,N}^T \end{bmatrix}$ is constructed, where N denotes the total number of samples that can be used, $N = t - p + 1$, and $U_{1,p,N}$ and $Y_{1,p,N}$ are block-Hankel matrices defined in terms of the input and output data, respectively, i.e.,

$$U_{1,p,N} = \begin{bmatrix} u_1 & u_2 & u_3 & \cdots & u_N \\ u_2 & u_3 & u_4 & \cdots & u_{N+1} \\ u_3 & u_4 & u_5 & \cdots & u_{N+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ u_p & u_{p+1} & u_{p+2} & \cdots & u_{N+p-1} \end{bmatrix},$$

and similarly for Y. Each u_i has m entries, and each y_i has ℓ entries. In both MOESP with past inputs and outputs and N4SID approaches, p is taken as $2s$, where s is the "number of block rows" considered for both the "past" or the "future" parts. A QR factorization, $H = QR$, is used for data compression, but the matrix Q is not needed subsequently; then a singular value decomposition (SVD) of a certain matrix reveals the order n of the system as the number of "non-zero" singular values. (For instance, the MOESP approach finds the SVD of the submatrix $R_{ms+1:(2m+\ell)s,(2m+\ell)s+1:2(m+\ell)s}$, while the N4SID approach is more involved, and it first finds an *oblique projection*.) System matrices are computed from the right singular vectors, and other submatrices of R. The covariance matrices are found using the residuals of a least squares problem. The Kalman gain is obtained by solving a discrete-time algebraic matrix Riccati equation.

Therefore, for both MOESP and N4SID approaches considered, the matrix whose triangular factor R is needed is the $N \times 2(m+\ell)s$ matrix $H = \begin{bmatrix} U^T_{1,2s,N} & Y^T_{1,2s,N} \end{bmatrix}$, where $N = t - 2s + 1$. The triangular factor itself is a $2(m+\ell)s \times 2(m+\ell)s$ matrix.

For sequential data processing, the QR factorization is done sequentially, by updating the upper triangular factor R. When all data have been compressed, the system order and system matrices are computed as in the previous case. For convenience, consider that there are only two data batches, denoted as (U_1, Y_1) and (U_2, Y_2), hence

$$U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}, \qquad Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix},$$

with the number of rows N_1 and N_2 (possibly the same), and let H_1 and H_2 be the block-Hankel-block matrices, corresponding to (U_1, Y_1) and (U_2, Y_2), respectively.

The algorithms outlined below are also summarized in [15, 17].

2.2 QR Factorization

The QR-based algorithms first compress the data (U_1, Y_1). One computes the QR factorization of H_1, $H_1 = Q_1 R_1$, and then the upper triangular factor is updated using a specialized QR factorization for the problem

$$\begin{bmatrix} R_1 \\ H_2 \end{bmatrix} = QR, \tag{2}$$

where the upper triangular structure of R_1 is exploited. Indeed, QR is a QR factorization of $H = \begin{bmatrix} H_1^T & H_2^T \end{bmatrix}^T$, as, with Q and Q_1 orthogonal matrices, we have

$$H^T H = H_1^T H_1 + H_2^T H_2 = R_1^T R_1 + H_2^T H_2 = R^T R.$$

The Q matrices are not needed in the subspace identification algorithms. The upper triangular factor R_1 is computed by a standard QR factorization, while for (2) one uses $2(m+\ell)s$ Householder transformations of the same order, $1+N_2$; each Householder transformation i annihilates all the elements of the i-th column of H_2, modifying the (i,i) element of R_1. If there are additional data batches, the same procedure is applied repeatedly for each new data batch.

Given the input-output data in a batch j, (U_j, Y_j), the computational routine has to build the appropriate segment H_j of the global block-Hankel-block matrix, H. This is done using a column-oriented process, exploiting the block-Hankel structure. The first (or single) segment H_1 is built either in the array `R`, which will store the triangular factor R, or in a workspace array, `W`, if `R` is not large enough. The remaining segments, if any, are built in the workspace, because `R` is anyhow needed for R. Efforts have been made to maximize the speed of the algorithmic process in the memory space provided. For instance, H_1 is constructed in `R`, if `R` is large enough. Otherwise, H_1 is constructed in `W`, its factor R_1 is computed in `W` and finally moved in `R`. If the workspace is not large enough to accommodate the entire data batch, then inner-loop sequential processing is additionally used, and the maximal number of rows that can be accommodated in the workspace are compressed in each inner cycle. Usually, the process could be more efficient when processing large data blocks at a time, due to the reduction in the overhead, but the optimal performance is, however, dependent on several factors, such as: use of an optimized BLAS library, cache memory size, block sizes for the underlying LAPACK block algorithms.

One complication appears when processing multiple data batches which are "connected", that is, which cannot be dealt with independently. When building the initial block-Hankel-block matrix H_1, the last row will have the index $N_1 - 2s + 1$, because, due to the Hankel structure, this row will include u_{N_1} and y_{N_1} as the last entries in the U_1 and Y_1 submatrices, respectively. A larger index cannot be used, as the corresponding entries will not be available in U_1 and Y_1. Therefore, the entries $u_{N_1-2s+2}, \ldots, u_{N_1}$ will not appear in the first column of H_1 for the N4SID algorithm. Similarly, the last $2s-1$ entries of U_2 will not appear in the first column of H_2. However, when the data batches are "connected", these last $2s-1$ elements in the current data batch should be added in front of the elements of the next data batch, in order to obtain the same triangular factor as when all batches would be processed in a single pass. Connecting the batches has been implemented by storing the $(m+\ell)(2s-1)$ "connection" elements in the workspace at the end of the processing phase of a batch (provided it is not the last), and restoring them at the beginning of the next phase. This way, all data are used and no information is omitted. No saving-restoring operations exist when the data batches are not connected.

2.3 Cholesky Factorization

In order to use the Cholesky factorization algorithm for data compression, one should first build the inter-correlation matrix $G = H^T H$, and then factor G, assuming it is positive definite (which is usually the case in practice, due to various noise components). For the N4SID approach, the block-Hankel matrix H_u, corresponding to the inputs, is

$$H_u = \begin{bmatrix} u_1^T & u_2^T & \cdots & u_{2s}^T \\ u_2^T & u_3^T & \cdots & u_{2s+1}^T \\ \vdots & \vdots & \vdots & \vdots \\ u_N^T & u_{N+1}^T & \cdots & u_{N+2s-1}^T \end{bmatrix}, \tag{3}$$

where $N = t - 2s + 1$; the block-Hankel matrix H_y, corresponding to the outputs, is written similarly, and $H = \begin{bmatrix} H_u & H_y \end{bmatrix}$. For the MOESP approach, the submatrix built from the last ms columns of H_u (the "future" inputs) comes in front of the submatrix built from the first ms columns of H_u (the "past" inputs part). Similarly, for identification with multiple batches, H_j is defined as H above, but with appropriate values for the inputs and outputs, as well as for the number of samples used.

The algorithm computes the symmetric (block) matrix $G = H^T H$ exploiting the block-Hankel structure. Let G_{uu}, G_{uy}, and G_{yy} denote the subblock-matrices of G corresponding to the input, input-output, and output (inter-)correlations, respectively, hence $G_{uu} = H_u^T H_u$, $G_{uy} = H_u^T H_y$, and $G_{yy} = H_y^T H_y$. Each subblock-matrix G_{uu}, G_{uy}, or G_{yy}, consists of $2s \times 2s$ submatrices of orders $m \times m$, $m \times \ell$, and $\ell \times \ell$, respectively. Denote $G_{uu}^{i,j}$ the (i,j)-th submatrix of G_{uu}, and similarly for G_{uy} and G_{yy}. The nice problem structure can be exploited [15, 17]. The formula for computing $G_{uu}^{i,j}$ can be written as follows

$$G_{uu}^{i,j} = \widehat{G}_{uu}^{i,j} + u_i u_j^T + u_{i+1} u_{j+1}^T + \cdots + u_{i+N-1} u_{j+N-1}^T, \tag{4}$$

where u_i^T is the i-th row of U, $j = 1{:}\,2s$, $i = 1{:}\,j$, and $\widehat{G}_{uu}^{i,j}$ is a zero matrix if the first (or single) data batch is computed, or it is the matrix $G_{uu}^{i,j}$ computed till the current batch, otherwise. Each matrix $G_{uu}^{j,j}$ is symmetric. The upper triangle of G_{uu} is computed or updated column-wise in the array `R`, that is, the block matrices are processed in the order $G_{uu}^{1,1}$, $G_{uu}^{1,2}$, $G_{uu}^{2,2}$, $\ldots$, $G_{uu}^{2s,2s}$. Only the submatrices of the first block-row are fully computed or updated with (4), i.e., (4) is used for $i = 1$ only. The remaining submatrices are determined exploiting the block-Hankel structure, using the updating formula, which follows from (4),

$$G_{uu}^{i+1,j+1} = \widehat{G}_{uu}^{i+1,j+1} - \widehat{G}_{uu}^{i,j} + G_{uu}^{i,j} + u_{i+N} u_{j+N}^T - u_i u_j^T, \tag{5}$$

applied for $j = 1{:}\,2s-1$, and for $i = 2{:}\,j$, for a given j; only the upper triangular part is evaluated for $i = j$. The submatrix $G_{uu}^{1,1}$ is computed using a BLAS 3 operation `SYRK`, the submatrices $G_{uu}^{1,2:2s}$ are computed using BLAS 3 operations `GEMM`, and the other submatrices are obtained using BLAS 2 and BLAS 1 updating operations `GERR` and `AXPY`, respectively.

The subblock-matrix G_{yy} is computed similarly. Part of the subblock-matrix G_{uy} is got using the formula

$$G_{uy}^{i,j} = \widehat{G}_{uy}^{i,j} + u_i y_j^T + u_{i+1} y_{j+1}^T + \cdots + u_{i+N-1} y_{j+N-1}^T, \tag{6}$$

where y_j^T is the j-th row of Y, and either $i = 1$ and $j = 1{:}\,2s$, or $i = 1{:}\,2s$ and $j = 1$, i.e., only the submatrices of the first block-row and block-column are fully computed (or updated) using (6). The remaining ones are determined exploiting the block-Hankel structure, using the updating formula

$$G_{uy}^{i+1,j+1} = \widehat{G}_{uy}^{i+1,j+1} - \widehat{G}_{uy}^{i,j} + G_{uy}^{i,j} + u_{i+N} y_{j+N}^T - u_i y_j^T, \tag{7}$$

for $j = 1{:}2s-1$, and $i = 1{:}2s-1$. The input-output correlations, G_{uy}, are computed (or updated) column-wise in the array `R`.

At the final batch for the MOESP approach, the past and future parts of the input submatrix H_u are interchanged after their computation. (Specifically, the upper triangular parts are interchanged, and the $(1,2)$ part is transposed in-situ.) This scheme is simpler and clearer than a directly implemented MOESP calculation.

Finally, the Cholesky factorization algorithm is applied on the computed correlation matrix G. In the rare case when this algorithm fails, the QR factorization is automatically used, for non-sequential processing. This is not possible for sequential processing because not all the data are available for the routine. The calculations could, however, be restarted under user's control.

The SLICOT toolbox includes an option for computing R by a fast QR algorithm, based on displacement rank techniques. The generators of the matrix $H^T H$ are computed and then used to obtain R. Details are given in [9]. The numerical properties and the efficiency of this algorithm are similar to those of the Cholesky factorization algorithm.

2.4 Computation of System Matrices

The other computational steps of the subspace-based identification procedures have been also analyzed for possible improvements, by exploiting any existing structure. For instance, an essential intermediate calculation in the N4SID algorithm is the determination of the so-called "oblique projection". Partition the factor R as $R = [\, U_p \;\; U_f \;\; Y_p \;\; Y_f \,]$, where the subscripts p and f stand for "past" and "future" data, respectively, and the four blocks have ms, ms, ℓs, and ℓs columns, respectively. Define $W_p = [\, U_p \;\; Y_p \,]$, and consider the residuals of the two least squares problems giving the oblique projection,

$$r_1 = W_p - U_f X_1, \qquad r_2 = Y_f - U_f X_2,$$

where X_1 and X_2 are the minimum norm least squares solutions of the problems

$$\min \|U_f X - W_p\|_2, \qquad \min \|U_f X - Y_f\|_2,$$

respectively. Then, the oblique projection can be computed as $O = r_2^T Q_1 Q_1^T$ [13, 14], where Q_1 consists of the first $k = \mathrm{rank}(r_1)$ columns of the orthogonal matrix in the QR factorization of r_1. No least squares problems should be actually solved. The advantage is that only orthogonal transformations are used for finding O, and therefore, the problem conditioning is preserved. Both problems have the same coefficient matrix, U_f, and it consists of two $ms \times ms$ submatrices, the second one being upper triangular; this structure is exploited.

A fast algorithm for computing the matrices B and D, using a structure exploiting QR factorization, has been described in [13, 14]. Essentially, the algorithm solves the following problem

$$\begin{bmatrix} Q_{1s} & Q_{1,s-1} & \cdots & Q_{12} & Q_{11} \\ 0 & Q_{1s} & \cdots & Q_{13} & Q_{12} \\ 0 & 0 & \cdots & Q_{14} & Q_{13} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & Q_{1s} \end{bmatrix} \begin{bmatrix} \Gamma_- & 0 \\ 0 & I_\ell \end{bmatrix} \begin{bmatrix} B \\ D \end{bmatrix} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ \vdots \\ K_s \end{bmatrix},$$

where Q_{1i} is $(\ell s - n) \times \ell$, for $i = 1\colon s$, Γ_- is $(\ell s - \ell) \times n$ (the leading part of the first n singular vectors), and K_i is $(\ell s - n) \times m$, $i = 1\colon s$. The first matrix product in the left-hand side is not computed, but the first matrix is fastly triangularized exploiting its structure. The matrices B and D are then found in two steps. Unfortunately, this fast algorithm finds least squares solutions in special cases only, namely, when the second matrix in the left hand side is square and nonsingular. The results obtained are often sufficiently accurate, especially for the MOESP technique. However, in some applications, the true least squares solutions should be computed using a less efficient algorithm, based on Kronecker products calculations; this algorithm is also available in the toolbox, and it solves a problem having half the size of the corresponding problem solved by the N4SID code. Extensive testing has shown that the use of the total least squares (TLS) solutions, as described in [13, 14], could provide less accurate results; consequently, the TLS strategy has been omitted in the toolbox.

The N4SID algorithm incorporated in the SLICOT toolbox differs from the published algorithm [19, 21] in many aspects and details. However, if few changes are made in the original algorithm, they will essentially produce the same matrices A, B, C, D; there could be differences in the signs of elements, but the systems computed by N4SID and SLICOT are related by a similarity transformation. The changes are:

2.4.1 SLICOT routines do not scale the input and output data by $1/\sqrt{t - 2s + 1}$; this scaling is actually needed only for computing the covariance matrices, when it can be performed in a faster way (on a part of the processed R factor).

2.4.2 The first n singular vectors are not scaled by the corresponding singular values.

2.4.3 The extended observability matrix Γ is not recomputed from the estimated A and C; extensive testing has shown that this strategy could sometimes slightly improve the accuracy, but equally well, diminish it.

An alternate algorithm is included for the computation of the matrices B and D. An extension and refinement of the method in [18, 24] is used. Specifically, denoting

$$X = \begin{bmatrix} \mathrm{vec}(D^T)^T & \mathrm{vec}(B)^T & x_0^T \end{bmatrix}^T,$$

where $\mathrm{vec}(M)$ is the vector obtained by stacking the columns of the matrix M, then X is the least squares solution of the system $SX = \mathrm{vec}(Y)$, with the matrix $S = \begin{bmatrix} \mathrm{diag}(U) & W \end{bmatrix}$, defined by

$$S = \left[\begin{array}{ccccc|c|c|c|c|c|c|c} U & & & & & & \cdots & & & \cdots & & \\ & U & & & & & \cdots & & & \cdots & & \\ & & \ddots & & & y^{11} & \cdots & y^{n1} & y^{12} & \cdots & y^{nm} & P\Gamma \\ & & & \ddots & & & \cdots & & & \cdots & & \\ & & & & U & & \cdots & & & \cdots & & \end{array}\right], \tag{8}$$

where $\mathrm{diag}(U)$ has ℓ block rows and columns. In this formula, y^{ij} are the outputs of the system for zero initial state, computed using the following model, for $j = 1\colon m$, and for $i = 1\colon n$,

$$\begin{aligned} x^{ij}(k+1) &= Ax^{ij}(k) + e_i u_j(k), \qquad x^{ij}(1) = 0, \\ y^{ij}(k) &= Cx^{ij}(k), \end{aligned} \tag{9}$$

where e_i is the i-th n-dimensional unit vector, Γ is given by

$$\Gamma = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{t-1} \end{bmatrix}, \tag{10}$$

and P is a permutation matrix that groups together the rows of Γ depending on the same row c_j of C, namely

$$\begin{bmatrix} c_j \\ c_jA \\ c_jA^2 \\ \vdots \\ c_jA^{t-1} \end{bmatrix},$$

for $j = 1\!:\ell$. The first block column, $\mathrm{diag}(U)$, is not explicitly constructed, but its structure is exploited. The last block column is evaluated using powers of A with exponents 2^k, but no nonnecessary powers are computed (ℓt is not expanded to a power of 2). No interchanges are applied. A special QR decomposition of the matrix S is computed. Let $U = q \begin{bmatrix} r^T & 0 \end{bmatrix}^T$ be the QR decomposition of U, if $m > 0$, where r is an $m \times m$ upper triangular matrix. Then, $\mathrm{diag}(q^T)$ is applied to W and $\mathrm{vec}(Y)$. The block-rows of S and $\mathrm{vec}(Y)$ are implicitly permuted so that the matrix S becomes

$$\begin{bmatrix} \mathrm{diag}(r) & W_1 \\ 0 & W_2 \end{bmatrix},$$

where W_1 has ℓm rows. Then, the QR decomposition of W_2 is computed (sequentially, if $m > 0$) and used to obtain B and x_0. The intermediate results and the QR decomposition of U are needed to find D. If a triangular factor is too ill conditioned, then singular value decomposition (SVD) is employed. SVD is not generally needed if the input sequence is sufficiently persistently exciting and t is large enough. If the matrix W cannot be stored in the workspace provided, the QR decompositions of W_2 and U are computed sequentially. The calculations simplify if D and/or x_0 are not needed.

The algorithm discussed above for computing the system matrices B and D, implemented in the SLICOT Library routine `IB01QD` (called by the driver `IB01CD`), is usually less efficient than the MOESP or N4SID algorithms implemented in SLICOT Library routine `IB01PD` (called by the driver `IB01BD`), because a large least squares problem has to be solved. The advantage is that the accuracy is better, since the computed matrices B and D are fitted to the input and output trajectories. However, if matrix A is unstable, the computed matrices B and D could be inaccurate.

When only x_0 should be determined, given the system matrices A, B, C, D, and the input-output sequences, a simpler method is implemented in the SLICOT Library routine `IB01RD` (also called by the driver `IB01CD`). This is an extension and refinement of the method in [24]. Specifically, the output $y_0(k)$ of the system for zero initial state is computed for $k = 1\!:t$ using the given model. Then, the following least squares problem is solved for x_0

$$\Gamma x_0 = \begin{bmatrix} y(1) - y_0(1) \\ y(2) - y_0(2) \\ \vdots \\ \vdots \\ y(t) - y_0(t) \end{bmatrix}. \tag{11}$$

The coefficient matrix Γ is evaluated using powers of A with exponents 2^k. The QR decomposition of Γ is computed. If its triangular factor R is too ill conditioned, then singular value decomposition of R is used. If the matrix Γ cannot be stored in the workspace provided, the QR decomposition is computed sequentially. Various particular cases are dealt with by special code portions, in order to increase the efficiency.

Structure exploiting algorithms and specialized linear algebra algorithms, performing all the processing steps of the MOESP and N4SID approaches, are used whenever possible. For instance, for computing the covariance matrices, the residuals of the linear least squares problem $MV = N$ should be computed, where

$$M = \begin{bmatrix} \begin{bmatrix} R_{11} \\ 0 \end{bmatrix} & Y \\ 0 & 0 \end{bmatrix}, \quad N = \begin{bmatrix} R_{1:3\widetilde{\ell},4} & X \end{bmatrix}, \tag{12}$$

the matrices M, N, Y and X have dimensions $((2m+\ell)s+\ell, ms+n)$, $((2m+\ell)s+\ell, \ell+n)$, $((2m+\ell)s, n)$, and $((2m+\ell)s+\ell, n)$, respectively, $R_{1:3\widetilde{\ell},4} = R_{1:(2m+\ell)s+\ell,(2m+\ell)s+1:(2m+\ell)s+\ell}$, and $R_{11} = R_{1:ms,1:ms}$ is upper triangular. The structure of this problem is fully exploited. Note also that the two block columns in M have been interchanged, compared to what would directly appear, in order to gain in efficiency. For the N4SID approach, the solution of this system is also needed, as it directly gives the system matrices A and C, and another matrix used for finding B and D.

3 SLICOT Identification Toolbox: Software Components

The SLICOT system identification toolbox consists in 13 user-callable and computational routines, three MATLAB interfaces (mexfiles), and over ten MATLAB m-files. The toolbox software components are also partly summarized in [16].

3.1 Fortran Routines

The Fortran routines are listed in Table 1, together with their function; the first three are driver routines. Together, these routines totalize about 0.4 Mb of source code.

Table 1: SLICOT system identification routines.

Routine	Function
IB01AD	preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system and finds an estimate of the system order. The input-output data can, optionally, be processed sequentially.
IB01BD	estimates the system matrices A, C, B, and D, the noise covariance matrices Q, R_y, and S, and the Kalman gain matrix K of a linear time-invariant state space model, using the processed triangular factor R of the concatenated block-Hankel matrices, provided by SLICOT Library routine IB01AD.
IB01CD	estimates the initial state and, optionally, the system matrices B and D of a linear time-invariant discrete-time system, given the system matrices (A, B, C, D), or only the matrix pair (A, C), respectively, and the input and output trajectories of the system.
IB01MD	computes the upper triangular factor in the QR factorization of the block-Hankel matrix built from input-output data for subspace identification.
IB01MY	computes the upper triangular factor in the QR factorization of the block-Hankel matrix, using a fast QR algorithm.
IB01ND	finds the singular value decomposition giving the system order, using the triangular factor of the concatenated block-Hankel matrices.
IB01OD	finds the system order, using the singular value decomposition.
IB01OY	asks for user's confirmation of the system order found.
IB01PD	estimates the system matrices and covariance matrices of a linear time-invariant state space model.
IB01PX	computes the matrices B and D of a linear time-invariant state space model using Kronecker products.
IB01PY	computes the triangular QR factor of a structured matrix and estimates the matrices B and D of a linear time-invariant state space model.
IB01QD	estimates the initial state and the system matrices B and D of a linear time-invariant state space model, given the matrix pair (A, C) and the input and output trajectories.
IB01RD	estimates the initial state of a linear time-invariant state space model, given the system matrices (A, B, C, D) and the input and output trajectories.

The calculations are organized so that much flexibility is achieved. For instance, it is possible to compute the A and C matrices via the MOESP approach, and the B and D matrices either via the N4SID approach, or by using the input-output data again (Subsection 2.4).

The design of these subroutines aimed to provide the user the most convenient and flexible way to perform the identification calculations. However, there is no unique driver to estimate the system matrices starting from the input-output records. The calculations have been split in two or three parts.

The *first part* compresses the input-output data and delivers a *processed* upper triangular factor R of the block-Hankel-block matrix H (in an array R), the singular values defining the system order, and an estimate of the system order, n. These calculations are performed by the driver `IB01AD`, which calls the computational routines `IB01MD`, `IB01ND`, `IB01OD`, and indirectly `IB01MY` (by `IB01MD`), and `IB01OY` (by `IB01OD`). The numerical values returned in the array `R` by the driver `IB01AD` do not represent the factor R, but an appropriate input for the next calculations. If the user actually needs R in a particular application, the lower-level computational routine `IB01MD` should be directly called. The available options of the driver `IB01AD`, which can be selected using character strings as actual arguments in the call,[2] are:

`METH` Specifies the subspace identification method to be used, as follows:
= `'M'`: MOESP algorithm with past inputs and outputs;
= `'N'`: N4SID algorithm.

`ALG` Specifies the algorithm for computing the triangular factor R, as follows:
= `'C'`: Cholesky algorithm applied to the correlation matrix of the input-output data;
= `'F'`: Fast QR algorithm;
= `'Q'`: QR algorithm applied to the concatenated block Hankel matrices.

`JOBD` Specifies whether or not the matrices B and D should later be computed using the MOESP approach, as follows:
= `'M'`: the matrices B and D should later be computed using the MOESP approach;
= `'N'`: the matrices B and D should not be computed using the MOESP approach.
This parameter is not relevant for `METH` = `'N'`.

`BATCH` Specifies whether or not sequential data processing is to be used, and, for sequential processing, whether or not the current data block is the first block, an intermediate block, or the last block, as follows:
= `'F'`: the first block in sequential data processing;
= `'I'`: an intermediate block in sequential data processing;
= `'L'`: the last block in sequential data processing;
= `'O'`: one block only (non-sequential data processing).
NOTE that when 100 cycles of sequential data processing are completed for `BATCH` = `'I'`, a warning is issued, to prevent for an infinite loop.

`CONCT` Specifies whether or not the successive data blocks in sequential data processing belong to a single experiment, as follows:
= `'C'`: the current data block is a continuation of the previous data block and/or it will be continued by the next data block;
= `'N'`: there is no connection between the current data block and the previous and/or the next ones.
This parameter is not used if `BATCH` = `'O'`.

`CTRL` Specifies whether or not the user's confirmation of the system order estimate is desired, as follows:
= `'C'`: user's confirmation;
= `'N'`: no confirmation.
If `CTRL` = `'C'`, a reverse communication routine, `IB01OY`, is indirectly called (by SLICOT

[2] One could use, for instance, `'MOESP'` for the parameter `METH`, to improve the readability of the code.

Library routine IB01OD), and, after inspecting the singular values and system order estimate, n, the user may accept n or set a new value. IB01OY is not called if CTRL = 'N'.

The reason for including the option JOBD is just to skip some computations when they are not necessary. The lower-level routine IB01OY is actually a simple template, which the user could call, or easily modify, in order to tailor it for a specific application. The other options are self-explanatory.

The *second part* uses the results delivered by IB01AD and computes the system matrices, (A, B, C, D), the covariance matrices (Q, R_y, S) (representing the state, output, and state-output (cross-)covariance matrices, respectively), and the Kalman gain matrix. These calculations are performed by the driver IB01BD, which calls the computational routine IB01PD, and this, in turn, calls IB01PX (for the N4SID approach) and IB01PY (for the standard MOESP approach). The available options of the driver IB01BD are:

METH Specifies the subspace identification method to be used, as follows:
= 'M': MOESP algorithm with past inputs and outputs;
= 'N': N4SID algorithm;
= 'C': combined method: MOESP algorithm for finding the matrices A and C, and N4SID algorithm for finding the matrices B and D.

JOB Specifies which matrices should be computed, as follows:
= 'A': compute all system matrices, A, B, C, and D;
= 'C': compute the matrices A and C only;
= 'B': compute the matrix B only;
= 'D': compute the matrices B and D only.

JOBCK Specifies whether or not the covariance matrices and the Kalman gain matrix are to be computed, as follows:
= 'C': the covariance matrices only should be computed;
= 'K': the covariance matrices and the Kalman gain matrix should be computed;
= 'N': the covariance matrices and the Kalman gain matrix should not be computed.

The ability to compute only part of the results, using the options JOB and JOBCK, is important in some applications. For instance, typical calculations performed by the Leuven Measurement Systems (LMS) International software, for mechanical systems, involve the computation of the matrices A and C for an increasing sequence of system orders; the poles (the eigenvalues of A) are computed for each system order, and *stabilization diagrams* are used to decide which poles actually correspond to physical poles, and only these poles are retained and used to build the final A and C matrices; then, the corresponding matrices B and D are computed using the approach implemented in the third SLICOT driver, IB01CD.

The *third part* uses the matrices A and C delivered by IB01BD and the input-output trajectories, $\{u_k\}$ and $\{y_k\}$, $k = 1{:}\,t$, and estimates the matrices B and D, and/or the initial state of the system, x_0, denoted as x(0) below. These calculations are performed by the driver IB01CD, which calls the computational routine IB01QD, if both the matrices B, D, and the initial state x_0 are desired, or IB01RD, if x_0 only is desired. For the second case, the matrices B and D should also be provided at input to the driver. The available options of the driver IB01CD are:

Table 2: Driver IB01CD: options, the data used, and the returned results.

JOBX0	COMUSE	JOB	Data used	Returned results
X	C	B	A, C, u, y	x, B
X	C	D	A, C, u, y	x, B, D
N	C	B	A, C, u, y	$x = 0, B$
N	C	D	A, C, u, y	$x = 0, B, D$
X	U	B	A, B, C, u, y	x
X	U	D	A, B, C, D, u, y	x
N	U	*	–	$x = 0$
X	N	*	A, C, y	x
N	N	*	–	–

`JOBX0` Specifies whether or not the initial state should be computed, as follows:
= 'X': compute the initial state x_0;
= 'N': do not compute the initial state (possibly, because x_0 is known to be zero).

`COMUSE` Specifies whether the system matrices B and D should be computed or used, as follows:
= 'C': compute the system matrices B and D, as specified by `JOB`;
= 'U': use the system matrices B and D, as specified by `JOB`;
= 'N': do not compute/use the matrices B and D.
If `JOBX0` = 'N' and `COMUSE` <> 'N', then x_0 is set to zero.
If `JOBX0` = 'N' and `COMUSE` = 'N', then x_0 is neither computed nor set to zero.

`JOB` If `COMUSE` = 'C' or 'U', specifies which of the system matrices B and D should be computed or used, as follows:
= 'B': compute/use the matrix B only (D is known to be zero);
= 'D': compute/use the matrices B and D.
The value of `JOB` is irrelevant if `COMUSE` = 'N' or if `JOBX0` = 'N' and `COMUSE` = 'U'.

The combinations of options, the data used, and the returned results, are given in Table 2, where '*' denotes an irrelevant value. For `JOBX0 = 'N'` and `COMUSE = 'N'`, the driver `IB01CD` does not perform any calculations, but just returns.

3.2 Interfaces: Mexfiles and m-files

The SLICOT system identification routines can be used from a MATLAB environment using the interfaces described in this section. The general framework of integrating the SLICOT Fortran routines in a MATLAB-like environment is described in [11]. Both mexfiles and m-files are provided in the SLIDENT toolbox. The mexfiles correspond to the Fortran drivers. Their interface is more complicated than that of the m-files, but they provide more flexibility and generality. The m-files call the mexfiles and are included for user's convenience. On line information about the mexfiles and m-files can be obtained in the usual manner, e.g., by typing `help sident`, for getting details about the mexfile `sident`. The same interfaces could also be used in a Scilab environment [4]. The mexfiles are listed in Table 3.

Table 3: SLICOT system identification: mexfile interfaces to MATLAB/Scilab.

Mexfile	Function
`order`	preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system and finds an estimate of the system order. The input-output data can, optionally, be processed sequentially.
`sident`	computes a state-space realization (A, B, C, D), the Kalman predictor gain K, and the covariances of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP, N4SID, or their combination).
`findBD`	estimates the initial state x_0 and/or the matrices B and D of a discrete-time system, given the system matrices A, C, and possibly B, D, and the input and output trajectories $\{u_k\}$, $\{y_k\}$, of the system.

The mexfiles and m-files include several options which could be of interest for the user; all these options have default values, and hence the calling statements could be very simple. One such option is `printw`, a switch for printing the warning messages; it should be set to 1 to print these messages, or to 0, otherwise (the default).

The calling sequences for the three mexfiles are shown below.

```
[R,n(,sval)(,rcnd)] = order(meth,alg,jobd,batch,conct,s,Y(,U,tol,
                            printw,ldwork,R));
[(A,C)(,B(,D))(,K(,Q,Ry,S))(,rcnd)] = sident(meth,job,s,n,l,R(,tol,t,
                                             A,C,printw));
[(x0)(,B(,D))(,V)(,rcnd)] = findBD(jobx0,comuse(,job),A(,B),C(,D),Y
                                   (,U,tol,printw,ldwork));
```

where the parameters put inside the brackets are optional, and their compulsory grouping is also indicated. Most of these parameters have clear meaning, or details have been already given in Subsection 3.1. The input data argument `U` is optional in the mexfiles and m-files which could involve it. The parameter `ldwork` in the mexfiles `order` and `findBD` allows the user to specify other sizes than the default ones for working arrays. Usually, the SLICOT mexfiles are using exactly the sizes needed for the arrays, but providing larger sizes for working arrays could improve the efficiency, e.g., by enabling the calls of the LAPACK block algorithms. The influence of the workspace length, when Fortran, not optimized BLAS routines are used, is shown in Figure 1 in Section 5. The parameter `rcnd` returns the reciprocal condition numbers, and, possibly, an error bound for the Riccati equation solution (needed for computing the Kalman gain); these accuracy indicators are useful in assessing the reliability of the computed results.

The basic MATLAB m-files which call the mexfiles are presented in Table 4. The *system object* defined in the MATLAB Control Toolbox is used, whenever possible.

Table 4: SLICOT system identification: m-file interfaces.

m-file	Function
`order`	help function for `order` mexfile.
`sident`	help function for `sident` mexfile.
`findBD`	help function for `findBD` mexfile.
`findR`	preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP or N4SID), and estimates the system order.
`findABCD`	finds the system matrices and the Kalman gain of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using MOESP, N4SID, or their combination.
`findAC`	finds the system matrices A and C of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using MOESP or N4SID.
`findBDK`	finds the system matrices B and D and the Kalman gain of a discrete-time system, given the system order, the matrices A and C, and the relevant part of the R factor of the concatenated block-Hankel matrices, using MOESP or N4SID.
`findx0BD`	estimates the initial state and/or the matrices B and D of a discrete-time linear system, given the (estimated) system matrices A, C, and a set of input/output data.
`inistate`	estimates the initial state of a discrete-time system, given the (estimated) system matrices, and a set of input/output data.

The calling sequences for the computational m-files (other than the three help files associated to the mexfiles) are listed below:

```
[R,n(,sval,rcnd)]       = findR(s,Y(,U,meth,alg,jobd,tol,printw));
[sys(,K,Q,Ry,S,rcnd)] = findABCD(s,n,l,R(,meth,nsmpl,tol,printw));
[A,C(,rcnd)]            = findAC(s,n,l,R(,meth,tol,printw));
[B(,D,K,Q,Ry,S,rcnd)] = findBDK(s,n,l,R,A,C(,meth,job,nsmpl,tol,printw));
[x0,B,D(,V,rcnd)]       = findx0BD(A,C,Y(,U,withx0,withd,tol,printw));
[x0(,V,rcnd)]           = inistate(sys,Y(,U,tol,printw));
```

The input parameter `tol` is a vector with two elements in case of `findR`, and a scalar, otherwise. `tol(1)` (or `tol`) is the tolerance for estimating the rank of matrices. If `tol(1)` > 0, the given value of `tol(1)` is used as a lower bound for the reciprocal condition number. `tol(2)` is the tolerance for estimating the system order. If `tol(2)` ≥ 0, the estimate is indicated by the index of the last singular value greater than or equal to `tol(2)`. (Singular values less than tol(2) are considered as zero.) When $\texttt{tol(2)} = 0$, then `s*eps*sval(1)` is used instead `tol(2)`, where `sval(1)` is the maximal singular value, and `eps` is the relative machine precision. When $\texttt{tol(2)} < 0$, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default values are: `tol(1) = prod(size(`*matrix*`))*eps`; $\texttt{tol(2)} = -1$.

Shorter calls are, of course, possible, e.g.,

```
[sys,rcnd] = findABCD(s,n,l,R);
[B,D,rcnd] = findBDK(s,n,l,R,A,C);
[B,D]      = findx0BD(A,C,Y,U,0);
x0         = inistate(A,C,Y);
x0         = inistate(A,B,C,Y,U);
```

The parameter `sys` is a discrete-time `ss` MATLAB system object, consisting in a state-space realization $\texttt{sys} = (A, B, C, D)$.

Example 3.1 Assuming `u`, `y` already available, the simple MATLAB session shown below

```
s = 21;  l = size(y,2);              % findR uses MOESP by default.
[R,n,sval] = findR(s,y,u);  meth = 3; % Get A,C by MOESP; B,D by N4SID.
[sys,rcnd] = findABCD(s,n,l,R,meth);

disp('Estimated system matrices')
[A,B,C,D] = ssdata(sys)

x0 = inistate(sys,u,y);        % Estimate x(0) using A,B,C,D, u, and y
x  = ltitr(A,B,u,x0);
ye = x * C.' + u * D.';        % Estimate y, using A,B,C,D, u, and x(0)

disp('Relative output error 1-norm and reciprocal condition numbers')
err = norm(y - ye,1)/norm(y,1)
rcnd
```

produced the following results, for some given $\{u\}$ and $\{y\}$ data sequences,

```
Estimated system matrices

A =
   9.3206e-01   2.9657e-01   3.5373e-02   1.5266e-02
  -1.9087e-01   9.2076e-01  -1.6224e-02   5.1043e-02
  -1.3172e-02   2.0301e-02   8.5844e-01  -4.5612e-01
  -1.5919e-03  -2.6322e-02   2.4633e-01   7.6515e-01

B =
   1.2245e+00   8.9248e-01
```

```
   7.7178e-01   4.5673e-01
  -4.1280e-01   4.1149e-01
   2.3463e-01  -1.2672e-01

C =
   2.2571e-01  -1.8491e-01  -3.5805e-01  -3.3156e-01
   2.1009e-01  -2.6185e-01   2.7019e-01   3.1076e-01

D =
   5.4624e-02   1.9648e-03
   2.7838e-03   3.7158e-02

Relative output error 1-norm and reciprocal condition numbers

err =
   1.3073e-06

rcnd =
   9.6651e-01
   1.0000e+00
   4.9869e-03
   3.9396e-01
```

□

Four additional method-oriented m-files, briefly described in Table 5, have been included in the SLIDENT toolbox.

Table 5: SLICOT system identification: method-oriented m-file interfaces.

m-file	Function
`slmoesp`	finds the system matrices and the Kalman gain of a discrete-time system, using MOESP subspace identification technique.
`sln4sid`	finds the system matrices and the Kalman gain of a discrete-time system, using N4SID subspace identification technique.
`slmoen4`	finds the system matrices and the Kalman gain of a discrete-time system, using combined MOESP and N4SID subspace identification techniques: the matrices A and C are found via MOESP, and B and D, via N4SID.
`slmoesm`	finds the system matrices, the Kalman gain, and the initial state of a discrete-time system, using combined MOESP subspace identification and system simulation techniques: the matrices A and C are found via MOESP, B and D via simulation.

The calling sequences for these method-oriented m-files are listed below.

```
[sys(,K,rcnd,R)]    = slmoesp(s,Y,U,n,alg,tol,printw);
[sys(,K,rcnd,R)]    = sln4sid(s,Y,U,n,alg,tol,printw);
[sys(,K,rcnd,R)]    = slmoen4(s,Y,U,n,alg,tol,printw);
[sys(,K,rcnd,x0,R)] = slmoesm(s,Y,U,n,alg,tol,printw);
```

If `n` $= 0$, or `n = []`, or `n` is omitted from the input parameters, the user is prompted to provide its value, after inspecting the singular values, shown as a bar plot. If `n` < 0, then `n` is determined automatically, according to `tol(2)`. The optional output parameter `R` returns the processed upper triangular factor R of the block-Hankel-block matrix H, built from the input-output data. It can be used for fast identification of systems of various orders, using, for instance, the following commands:

```
[sys,K,rcnd,R] = sln4sid(s,Y,U,n0,alg,tol,printw);
for n = n0+1 : min( n0+nf, s-1 )
   [sys,K,rcnd] = sln4sid(s,Y,U,n,R,tol,printw);
end
```

The data values for `Y` and `U` are not used inside the loop (only the size of `Y` is needed), but `R` replaces `alg`. Clearly, the systems of various orders (from `n0+1` to min(`n0+nf, s-1`)), should be used inside the loop. The elements `rcnd(1)` and `rcnd(2)` are set to 1 when `sln4sid` is called with `R` instead of `alg`.

Shorter calls are, of course, possible, e.g.,

```
[sys,K] = slmoesp(s,Y);
[sys,K] = slmoesp(s,Y,[],n);
sys     = slmoesp(s,Y,U);
```

and similarly for the other m-files. The first two calls estimate the matrices A, C, and K of a stochastic system (with no inputs), for an order n found automatically, or prespecified, respectively.

The same numerical results as those obtained using `findR` and `findABCD` in Example 3.1 can be obtained using the following MATLAB code

```
s = 21;
sys = slmoen4(s,y,u);              % Get A,C by MOESP; B,D by N4SID.
```

4 Benchmarks for Identification

This topic is fully described in another NICONET report, entitled *Benchmarks for Identification*, available through the Niconet web page (`Reports` link), or by anonymous ftp from `wgs.esat.kuleuven.ac.be`, file `pub/WGS/REPORTS/nic1999-19.ps.Z`. This report also describes some rules for benchmarking in system identification. A valuable, publicly available collection of test data is the Database for Identification of Systems (DAISY), which is accessible from the website `wgs.esat.kuleuven.ac.be/sista/daisy`. This collection contains several datasets:

process industry systems, mechanical systems, biomedical systems, environmental systems, thermic systems, simulators, and time series. Some of the test data from DAISY have been used for testing the identification toolbox SLIDENT (see Section 5).

5 Industrial Examples

Besides the industrial examples from the DAISY data collection, several input-output sequences from the LMS International have been used to test the components of the SLIDENT toolbox. An effort is underway at LMS to incorporate the SLICOT routines for identification into their interactive software. The results obtained on some of the LMS industrial data will be presented in another Niconet report.

The rest of this section summarizes typical results obtained using the subspace-based techniques available in the SLIDENT toolbox. The data used is publicly available on the DAISY internet site:

`http://www.esat.kuleuven.ac.be/sista/daisy`

in order to increase the accessibility and the reproducibility. Specifically, Table 6 gives a summary description of the applications considered in this report, indicating the number of inputs m, outputs ℓ, block rows s, and data samples used t.

Table 6: Summary description of applications.

#	Application	m	ℓ	s	t
1	Glass tubes	2	2	20	1401
2	Labo Dryer	1	1	15	1000
3	Glass Oven	3	6	10	1247
4	Mechanical flutter	1	1	20	1024
5	Flexible robot arm	1	1	20	1024
6	Evaporator	3	3	10	6305
7	CD player arm	2	2	15	2048
8	Ball and beam	1	1	20	1000
9	Wall temperature	2	1	20	1680

Only one data batch has been used in all the tests whose results are given below. The numerical results have been obtained on either an IBM PC computer at 500 MHz, with 128 Mb memory, using Digital Visual Fortran V5.0, or on a Sun 4 sparc Ultra-2 computer, using operating system OS 5.6, and Sun WorkShop Compiler FORTRAN 77 5.0. In either cases, MATLAB 5.3.0.10183 (R11) has been employed.

In [15], results obtained on the PC computer for the data compressing step using the QR factorization and the fast Cholesky factorization algorithms (for the MOESP approach) are presented and compared with the MATLAB codes. Additional results are given in [9], using the fast algorithm exploiting the displacement structure. Both fast algorithms could be used for all applications, except Application 4, for which the computed correlation matrix was not positive definite. Compared to the standard QR factorization, the gain in efficiency was remarkable (usually, more than 50 times faster), and the accuracy was comparable. On the other hand, MATLAB calculations of the correlations appeared as rather inefficient and they are not included in the comparison. The results in [15] have also shown that MATLAB QR algorithm was faster than SLICOT calculations. The reason discovered later was the bad use of the computer cache memory in the SLICOT tests, correlated to the use of Fortran BLAS libraries, rather than of optimized BLAS. By reducing the size of the provided workspace, and implicitly exploiting the abilities of SLICOT routines to process the data sequentially (in either outer on inner cycles), the efficiency of SLICOT calculations has been improved. Timing results for various workspace sizes (on the PC) are shown in Figure 1. The notation "min" and "full" denote that the minimum amount and the recommended (assuming a large enough cache size) amount, respectively, of workspace needed for each application has been specified as the workspace size, while "full/c" means that the "recommended" workspace size has been divided by the scalar c. The value "full/24" provided a much better use of the cache memory for the most time consuming applications, and enabled to slightly outperform the MATLAB calculations. Since the efficiency of some calculations could depend on the size of the workspace provided, two of the mexfiles in the SLIDENT toolbox, `order` and `findBD`, include the parameter `ldwork`—the length of the working array. A default value is used when `ldwork` is not specified, or if it is less than the minimal workspace needed.

Table 7: Relative output errors using the QR or Cholesky factorization algorithms.

#	Relative output errors			
	SLmoesp	OMOESP	MOESP, SLmoesm	N4SID, SLn4sid
1	5.39e-01	1.02e+01	4.86e-01	6.09e-01
2	3.30e-02	7.33e-02	1.50e-02	2.16e-02
3	6.03e-01	6.42e-01	4.96e-01	6.21e-01
4	2.24e+02	6.49e+10	2.37e-01	2.94e-01
5	1.14e-01	7.38e+04	3.60e-02	4.51e-02
6	5.50e-01	5.67e-01	4.89e-01	5.53e-01
7	1.13e+02	8.06e+04	1.76e-01	3.45e-01
8	3.61e-01	2.21e+01	2.54e-01	2.78e-01
9	4.23e-01	4.20e-01	1.38e-01	1.38e-01

Table 7 presents comparative results, obtained on the PC, for the accuracy of the calculations using SLICOT algorithms, as well as the available MATLAB codes. Standard QR factorization has been used in all these calculations, but exactly the same accuracy results have been obtained using the Cholesky factorization algorithm, except for Application # 4, for which the Cholesky algorithm failed, and the QR algorithm was automatically called. The input and output trajectories for Application # 4 are plotted in the Figure 2 and Figure 3, respectively. The reported relative output error has been computed with the following MATLAB formula

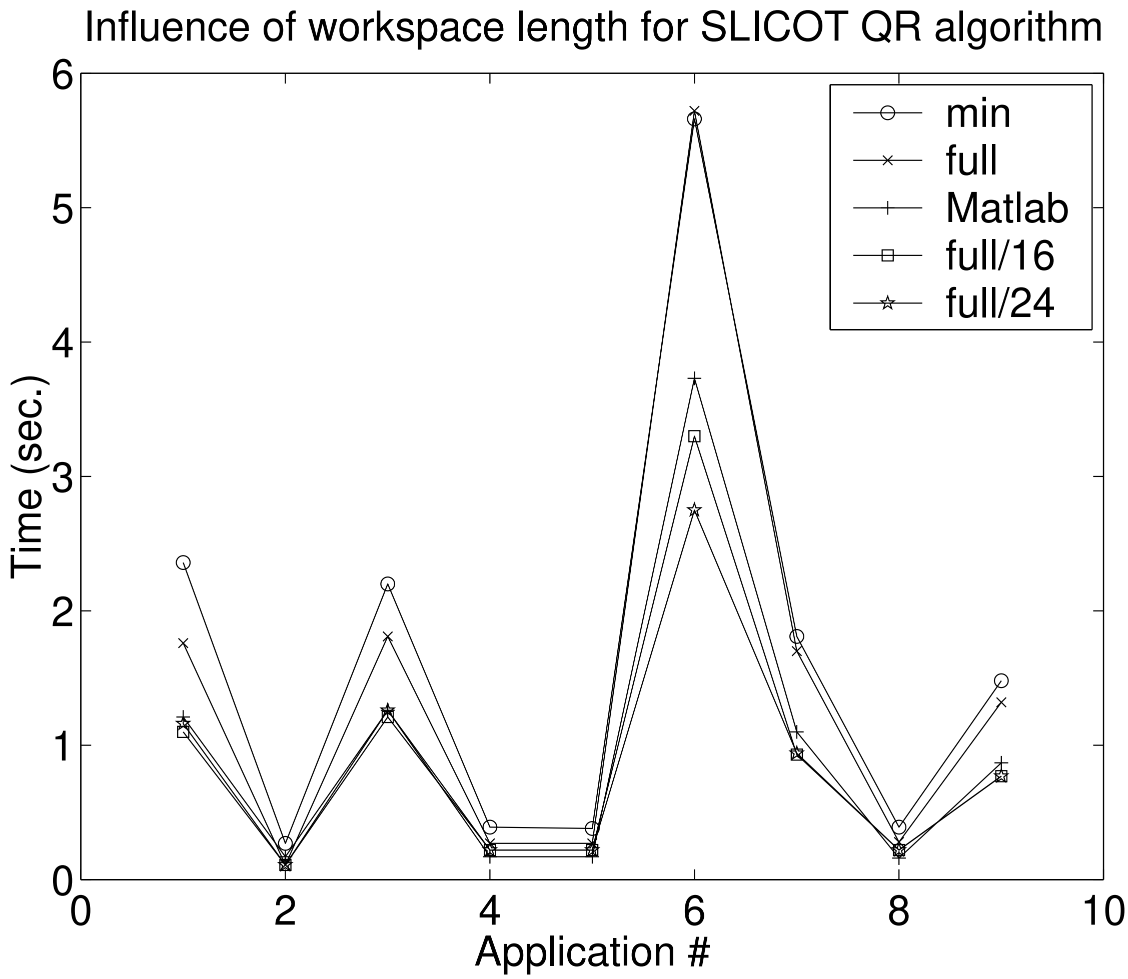

Figure 1: Influence of workspace length for the SLICOT QR factorization algorithm with non-optimized BLAS (MOESP approach).

```
err = norm(y - ye,1)/norm(y,1);
```

where `ye` denotes the estimated output of the system, evaluated using the estimated system matrices, A, B, C, and D, the given input trajectory, $\{u_k\}$, and an estimate of the initial state of the system, found, in turn, using A, B, C, D, and the given trajectories, $\{u_k\}$ and $\{y_k\}$. In the tables and figures, SLmoesp and SLn4sid denote the SLICOT implementations of the standard MOESP and N4SID approaches, OMOESP denotes a previous MATLAB code using TLS [23], MOESP denotes a new MATLAB code which computes the B and D matrices by solving a large least squares problem built using the matrix pair (A, C) and the input and output trajectories of the system [8], and N4SID denotes the robust MATLAB code in [21]. As already shown (Subsection 2.4), the SLICOT toolbox also contains the algorithm in the new MOESP code (called here SLmoesm, corresponding to the m-file `slmoesm`, or to the combination of the computational routines `IB01PD` and `IB01QD`, called by the driver routines `IB01BD` and `IB01CD`,

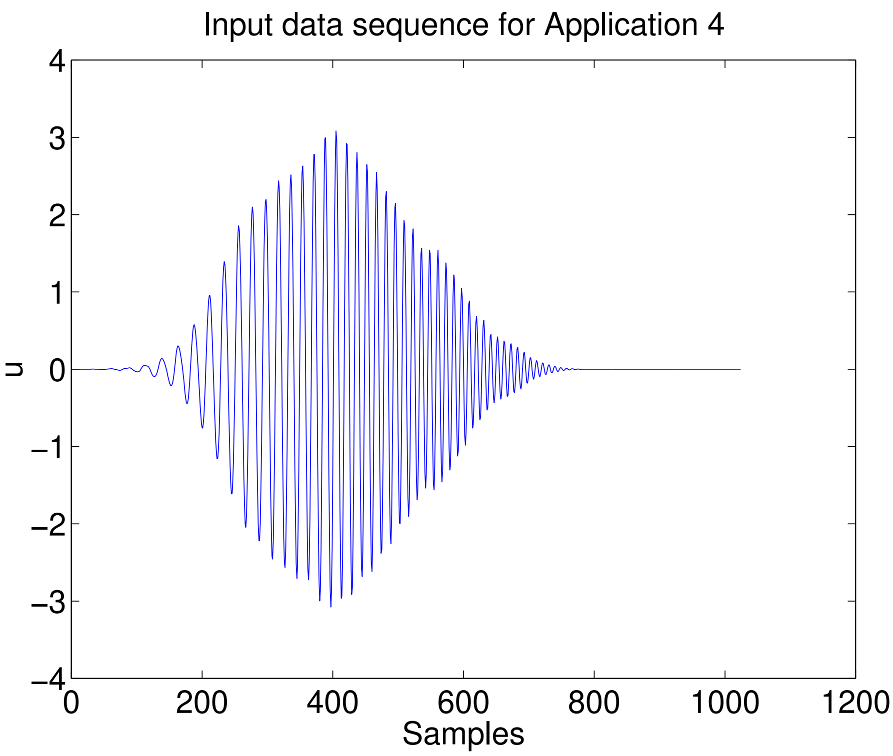

Figure 2: Input data sequence for Application # 4.

respectively); this algorithm gave the same accuracy as MOESP. The applications 4 and 7 are difficult for standard MOESP approach, because ill-conditioned matrices appear during the computation of B and D. However, there is no difficulty in solving these applications using the other methods, implemented in the m-files `sln4sid`, `slmoen4`, or `slmoesm`. Figure 4 plots the trajectories of the output, $\{y\}$, estimated output, $\{y_e\}$, and filtered output, $\{y_f\}$ (with Kalman gain also included in the estimated output calculations), when `slmoesm` was used for Application # 4. The filtered output is practically indistinguishable from the output.

The bar charts in Figure 5 illustrate the timing results on the Sun machine for computing system matrices using the QR factorization algorithm. Figure 6 presents the timing results when SLICOT routines used the Cholesky factorization algorithm. Since the bars in Figure 6 for SLICOT calculations have heights almost negligible compared to the MATLAB codes (except for Application 4), Table 8 presents the timing results numerically. When successful, the Cholesky factorization algorithm enabled to significantly increase the efficiency, while preserving the same accuracy as for the QR factorization algorithm. The speed-up factors vary between 10 and 20 when comparing to the SLICOT QR factorization algorithm, and between 15 and 40 when comparing to MATLAB codes. For a PC machine, the gain in efficiency was even larger. Figure 7 illustrates the timing results on the PC machine for computing the system matrices and Kalman gain using the Cholesky factorization algorithm, for all method-oriented m-files. For some applications and methods, the CPU time was below the measurable time (0.05 sec.), and in such cases the corresponding bars could not be plotted. It is apparent from Figure 7 that `slmoesp` and

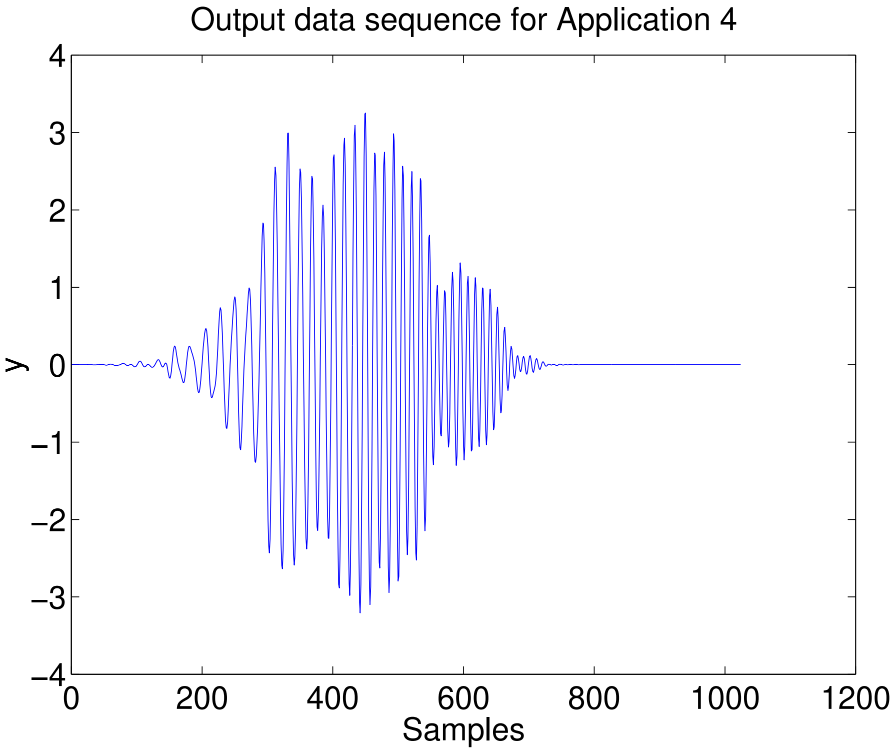

Figure 3: Output data sequence for Application # 4.

`slmoen4` were the most efficient methods, while `slmoesm` was less efficient for some applications, like the applications # 1, 3, 6, and 7.

6 Conclusions

Algorithmic, implementation and numerical details concerning subspace-based techniques for linear multivariable system identification have been described and compared. The techniques are implemented in the new system identification toolbox for the SLICOT Library—SLIDENT. This toolbox includes interfaces (mexfiles and m-files) to the MATLAB and Scilab environments, which improve the user-friendliness of the collection. The results obtained show that the fast algorithmic variants included in the toolbox can frequently be used, and they are significantly more efficient than the standard QR factorization and the MATLAB codes.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide: Second Edition*. SIAM, Philadelphia, 1995.

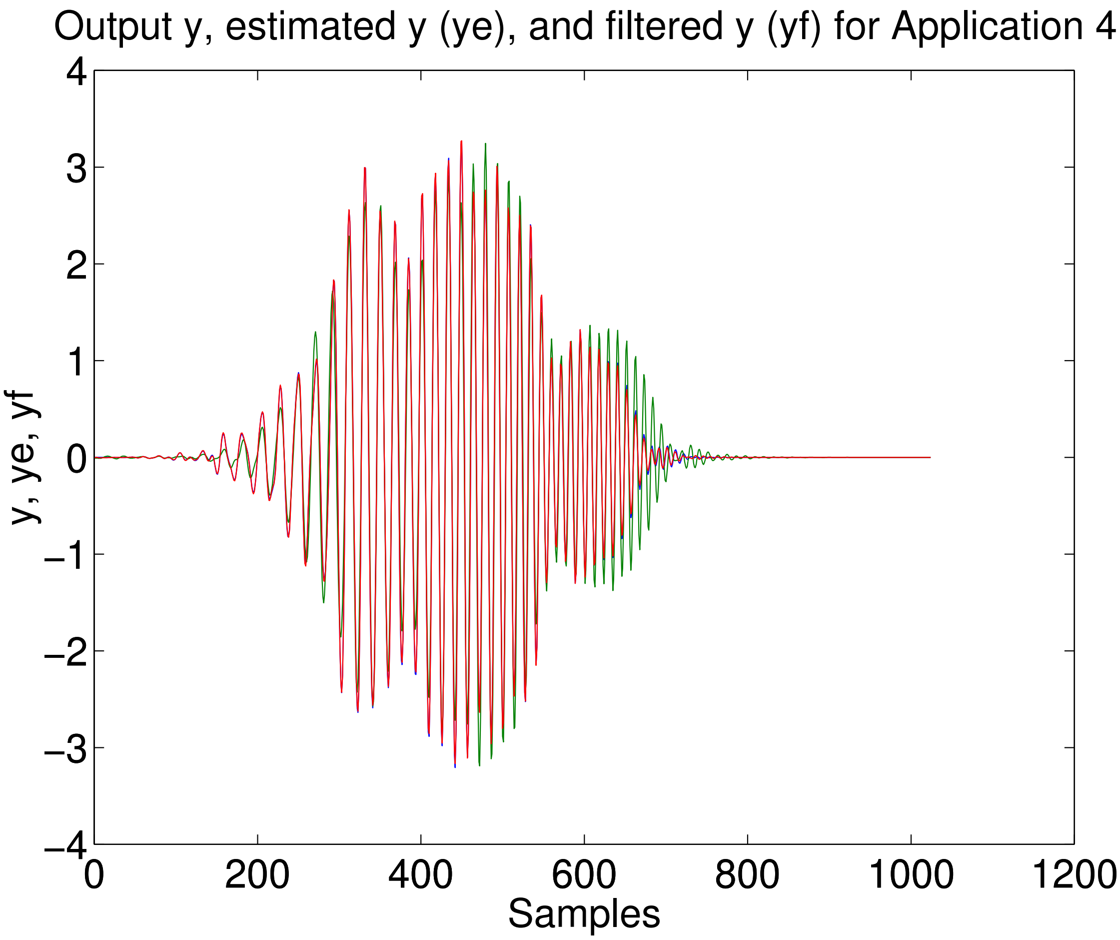

Figure 4: Output y, estimated y (`ye`), and filtered y (`yf`) for Application 4, using `slmoesm` m-file.

[2] 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, editor, *Applied and Computational Control, Signals, and Circuits*, volume 1, chapter 10, pages 499–539. Birkhäuser, 1999.

[3] B. De Moor, P. Van Overschee, and W. Favoreel. Numerical algorithms for subspace state-space system identification: An overview. *Applied and Computational Control, Signals, and Circuits*, 1, chapter 6:247–311, 1999.

[4] F. Delebecque and S. Steer. *Integrated Scientific Computing with Scilab.* Birkhäuser, Boston, 1997.

[5] 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.

[6] 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.

Table 8: CPU time (sec.) for computing the system matrices using Cholesky factorization algorithm in the SLICOT-based calculations.

#	Time				
	SLmoesp	OMOESP	MOESP	SLn4sid	N4SID
1	0.04	0.76	1.32	0.06	0.92
2	0.01	0.38	0.33	0.03	0.41
3	0.27	3.46	4.10	0.36	3.48
4	0.37	0.71	0.55	0.37	0.68
5	0.03	0.72	0.51	0.04	0.69
6	0.32	8.23	10.29	0.37	8.13
7	0.10	2.83	3.37	0.16	2.80
8	0.03	0.69	0.48	0.04	0.62
9	0.08	2.13	1.75	0.13	2.10

[7] B. Haverkamp. Efficient implementation of subspace method identification algorithms. Technical Report nic1999-3, Katholieke Universiteit Leuven (ESAT/SISTA), Leuven, Belgium, Jan. 1999. Available from `ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/`, file `SLWN1999-3.ps.Z`.

[8] B. Haverkamp and M. Verhaegen. SMI Toolbox: State space model identification software for multivariable dynamical systems. Report TUD/ET/SCE96.015, Delft University of Technology, Delft, The Nederlands, 1997. `http://lcewww.et.tudelft.nl/ haver/smi.html`.

[9] D. Kressner, N. Mastronardi, V. Sima, P. Van Dooren, and S. Van Huffel. A fast algorithm for subspace state-space system identification via exploitation of the displacement structure. *J. Comput. Appl. Math.*, 2000. Submitted for a special issue of the journal.

[10] 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.

[11] V. Mehrmann, V. Sima, A. Varga, and H. Xu. A MATLAB MEX-file environment of SLICOT. Technical Report SLWN1999-11, Katholieke Universiteit Leuven (ESAT/SISTA), Leuven, Belgium, Aug. 1999. Available from `ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/`, file `SLWN1999-11.ps.Z`.

[12] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle. On- and off-line identification of linear state space models. *Int. J. Control*, 49(1):219–232, 1989.

[13] V. Sima. Algorithms and LAPACK-based software for subspace identification. In *Proceedings of The 1996 IEEE International Symposium on Computer-Aided Control System Design, September 15–18, 1996, Ritz-Carlton, Dearborn, Michigan, U.S.A.*, pages 182–187, 1996. A compressed postscript file is available at the ftp address `ftp://wgs.esat.kuleuven.ac.be/pub/SISTA/vanhuffel/reports/Sima96.ps.Z`.

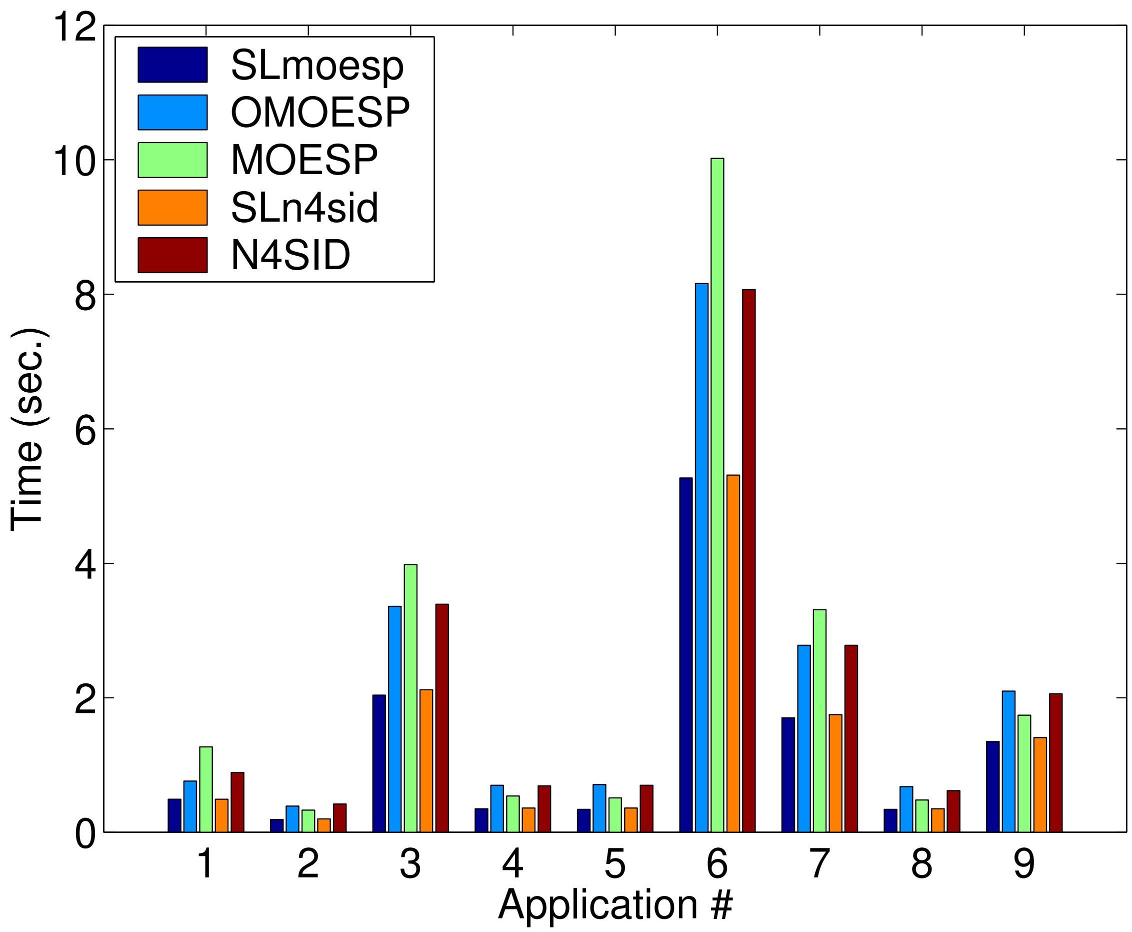

Figure 5: CPU time (sec.) for computing the system matrices using the QR factorization algorithm.

[14] V. Sima. Subspace-based algorithms for multivariable system identification. *Studies in Informatics and Control*, 5(4):335–344, 1996.

[15] V. Sima. Cholesky or QR factorization for data compression in subspace-based identification ? In *Proceedings of the Second NICONET Workshop on "Numerical Control Software: SLICOT, a Useful Tool in Industry", December 3, 1999, INRIA Rocquencourt, France*, pages 75–80, 1999. Organised by Numerics in Control Network (NICONET), BRITE/EURAM III BRRT–CT97–5040.

[16] V. Sima and S. Van Huffel. SLICOT subspace identification toolbox. In *Proceedings CD of the U.K. Automatic Control Conference, UKACC 2000, Cambridge, United Kingdom, September 4-7, 2000*, 2000. To appear.

[17] V. Sima and S. Van Huffel. Efficient numerical algorithms and software for subspace-based system identification. In *Proceedings of The IEEE International Conference on Control Applications and IEEE International Symposium on Computer-Aided Control Systems De-*

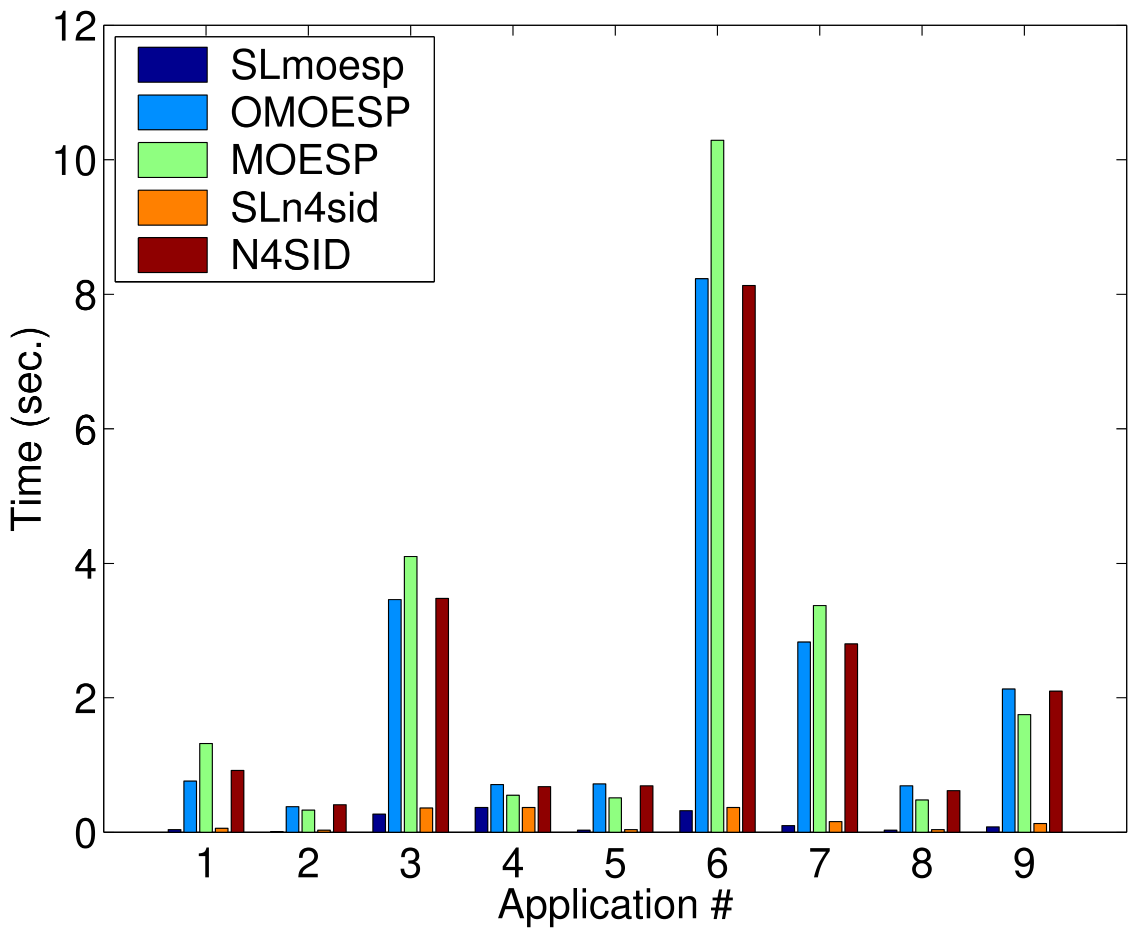

Figure 6: CPU time (sec.) for computing the system matrices using Cholesky factorization algorithm in the SLICOT-based calculations.

sign, September 25–27, 2000, Anchorage Hilton, Anchorage, Alaska, U.S.A., Sept. 2000. Omnipress. To appear.

[18] V. Sima and A. Varga. RASP-IDENT : Subspace model identification programs. Technical Report TR R888–94, Deutsche Forschungsanstalt für Luft- und Raumfahrt e. V., DLR – Oberpfaffenhofen, Institut für Robotik & Systemdynamik, Entwurfsorientierte Regelungstechnik (Prof. G. Grübel), D–82230 Weßling, Oct. 1994. 78 pages.

[19] P. Van Overschee. *Subspace Identification : Theory – Implementation – Applications*. PhD thesis, Katholieke Universiteit Leuven, Leuven, 1995.

[20] P. Van Overschee and B. De Moor. N4SID: Two subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1):75–93, 1994.

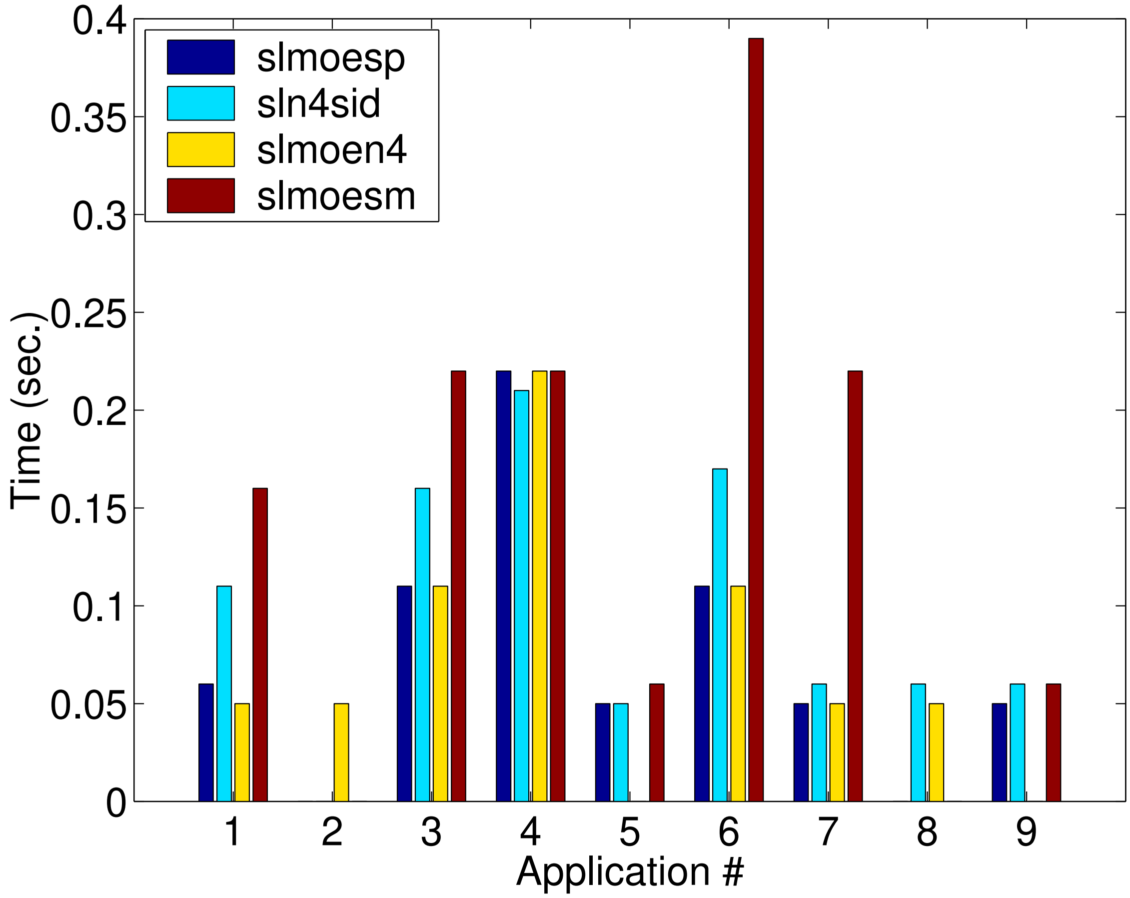

Figure 7: CPU time (sec.) for computing the system matrices and Kalman gain using the Cholesky factorization algorithm and the method-oriented m-files.

[21] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems : Theory – Implementation – Applications.* Kluwer Academic Publishers, Boston/London/Dordrecht, 1996.

[22] M. Verhaegen. Subspace model identification. Part 3: Analysis of the ordinary output-error state-space model identification algorithm. *Int. J. Control*, 58(3):555–586, 1993.

[23] M. Verhaegen. State space model identification toolbox. Technical Report, Delft University of Technology, Delft, Nov. 1993.

[24] M. Verhaegen and A. Varga. Some experience with the MOESP class of subspace model identification methods in identifying the BO105 helicopter. Technical Report TR R165-94, DLR Oberpfaffenhofen, Institute for Robotics and System Dynamics, D–82230 Wessling, June 1994.